



**Estratégia**  
CONCURSOS

## Linguagem SQL

Análise de Informações p/ TCU 2017 - Todos os cargos (Com videoaulas)

Professor: Thiago Rodrigues Cavalcanti

## AULA Extra: Linguagem de consulta estruturada (SQL).

### Sumário

Conceitos de SQL .....	2
1. Introdução a SQL .....	2
1.1. Linguagem procedural.....	5
1.2. DDL.....	6
1.2.1. Trabalhando com tabelas.....	7
1.2.2. Tipos de dados .....	8
1.2.2.1. CHARACTER.....	8
1.2.2.2. Binary String .....	10
1.2.2.3. Tipos numéricos.....	11
1.2.2.4. Datetime e interval.....	12
1.2.2.5. Boolean e XML.....	13
1.2.2.6. Tipos ROW, Collections e UDT.....	13
1.1.1. Restrições de integridade.....	15
1.2. Entendendo o catálogo SQL - (INFORMATION_SCHEMA) .....	18
1.2.1. Information Schema.....	21
1.3. DML.....	21
1.3.1. O COMANDO SELECT .....	21
1.3.2. ALIAS, DISTINCT e operadores .....	23
1.3.3. Consultas aninhadas (IN, EXISTS, ALL, SOME, ANY).....	26
1.3.4. Funções agregadas (GROUP BY e HAVING).....	27
1.3.5. Operações de conjuntos .....	28
1.3.6. Junção.....	29
1.3.7. INSERT, UPDATE e DELETE.....	30
1.4. DDL Complementos: VIEW .....	31
1.5. SQL Embutido .....	33
1.6. Transações em SQL.....	34
1.7. Index (DDL).....	37
1.8. Extensões procedurais .....	38
1.9. Segurança – Utilizando a DCL.....	42
1.9.1. A hierarquia do privilégio de acesso .....	43
1.9.2. Garantindo privilégios aos usuários .....	43
1.9.3. Roles.....	45
1.9.4. WITH GRANT OPTION.....	45
1.9.5. Removendo privilégios .....	46
2. Interfaces de utilização .....	47
2.1. ODBC .....	48
2.2. Arquitetura ODBC.....	49
Questões Comentadas.....	51
Considerações finais.....	108
Referências .....	108

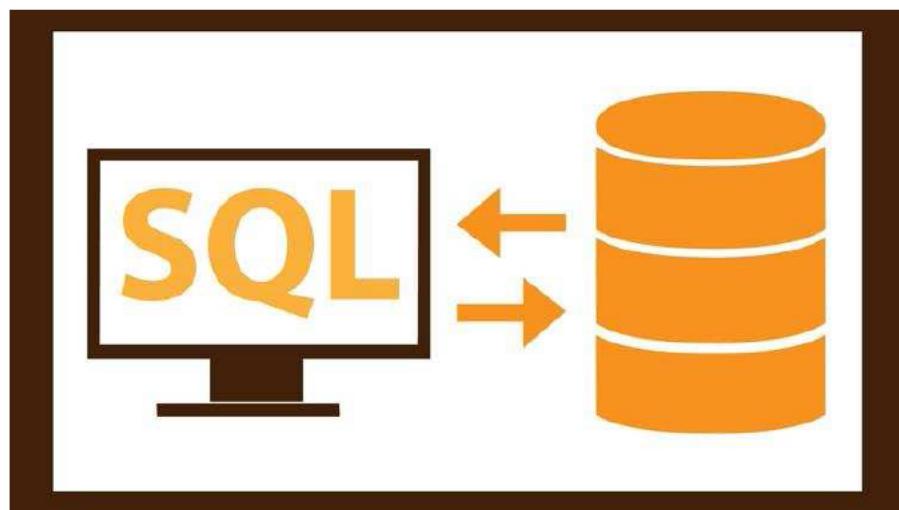
## Conceitos de SQL

Esta aula tem como conteúdo programático a linguagem SQL, mais especificamente o padrão SQL ANSI. Espero que gostem! Dúvidas e sugestões podem ser enviadas para mim por e-mail ou pelos canais de comunicação do Estratégia Concursos.

Dentro do contexto da Linguagem SQL vamos apresentar as construções e conceitos fundamentais. Vamos entender como as bancas abordam este assunto. SQL é estatisticamente o assunto mais cobrado dentro do rol de conteúdos de banco de dado nas provas de concurso. Vamos em frente!

### 1. Introdução a SQL

A linguagem SQL foi criada por um grupo de pesquisadores do laboratório da IBM em San Jose, Califórnia, mais especificamente pelos pesquisadores Donald D. Chamberlin e Reymond Boyce. Eles publicaram um artigo denominado “SEQUEL: A Structured English Query Language” no qual apresentavam detalhes da linguagem como sua sintaxe e operações. Alguns anos depois da publicação do artigo, devido a um conflito de patentes com um projeto secreto de uma empresa inglesa de aviação, SEQUEL mudou de nome para SQL.



A Structured Query Language – SQL – pode ser considerada uma das principais razões para o sucesso dos bancos de dados relacionais. Ela se provou bastante útil e foi adotada como padrão pela ANSI (American National Standards Institute). São várias versões da norma publicadas ao longo do tempo: 1986, 1989, 1992, 1999, 2003, 2006, 2008, 2011. Nossa foco de estudo se baseia na parte da norma que apresenta os fundamentos de SQL. Mas como podemos definir a linguagem?

Primeiramente, SQL é uma linguagem de programação reconhecida internacionalmente e usadas para definição e manutenção de bandos de dados

relacionais. A principal característica da linguagem é ser **declarativa**, ou seja, os detalhes de implementação dos comandos são deixados para os SGBDs relacionais.

Outro aspecto importante é que SQL não existe fora do contexto relacional devido ao fato de ser fundamentada no modelo relacional. A linguagem utiliza a álgebra relacional como base teórica para implementar as operações dentro dos SGBDs.

Quando começamos a estudar a linguagem precisamos entender as subdivisões que existem dentro dos comandos possíveis. Alguns autores chegam a dividir a linguagem em cinco categorias. As categorias são baseadas nas funcionalidades que cada comando executa. Vejam a lista abaixo:



- **DDL** – Data Definition Language – A linguagem de definição de dados contém comandos que criam, modificam e excluem objetos de banco de dados. São exemplos de comando: CREATE, ALTER, DROP e TRUNCATE.
- **DML** – Data Manipulation Language – A linguagem de manipulação de dados fornece instruções para trabalhar como os dados armazenados como SELECT, INSERT, UPDATE e DELETE.
- **DQL** – Data Query Language – A linguagem de consulta de dados é um subconjunto da DML que possui apenas a instrução de SELECT.
- **DTL** – Data Transaction Language – Linguagem de transação de dados inclui comandos de COMMIT, ROLLBACK e SAVEPOINT
- **DCL** – Data Control Language – A linguagem de controle de dados contém os comandos relacionados com as permissões de controle de acesso. Garante os privilégios aos usuários para acessar os objetos do banco. Os mais conhecidos comandos são o GRANT e o REVOKE.

Agora que já entendemos a diferença entre cada uma das subdivisões da linguagem vamos fazer duas questões para fixação.



#### 01. BANCA: FCC ANO: 2015 ÓRGÃO: MANAUSPREV PROVA: ANALISTA PREVIDENCIÁRIO - TECNOLOGIA DA INFORMAÇÃO

A linguagem SQL é dividida em subconjuntos de acordo com as operações que se deseja efetuar sobre um banco de dados. Considere os grupos de comandos:  
I. CREATE, ALTER, DROP.

- II. GRANT, REVOKE.  
III. DELETE, UPDATE, INSERT.

Os comandos listados em

- A) I correspondem à Data Control Language – DCL e II à Data Definition Language – DDL.  
B) I correspondem à Data Manipulation Language – DML e III à Data Control Language – DCL.  
C) II correspondem à Data Manipulation Language – DML e III à Data Control Language – DCL.  
D) I correspondem à Data Definition Language – DDL e III à Data Manipulation Language – DML.  
E) II correspondem à Data Control Language – DCL e III à Data Definition Language – DDL.

**Comentário:** Observe que pelo que acabamos de expor sobre as sobre os subconjuntos da linguagem temos, no item I comandos de DDL, no item II comandos de DCL – Data control language e, no item III comandos de DML. Este fato confirma nossa resposta na alternativa D.

**Gabarito D.**



**02. Ano: 2016 Banca: FCC Órgão: TRE-SP Cargo: Analista Judiciário de TI – Q. 56.**

Em uma situação hipotética, ao ser designada para atender aos requisitos de negócio de um usuário, uma Analista de Sistemas do TRE-SP escreveu expressões e comandos para serem executados em um Banco de Dados Relacional que visavam (1) criar uma tabela que contivesse dados de processos partidários, (2) controlar a segurança e o acesso a ela e (3) manipular dados nela. Desta forma ela, se valeu, correta e respectivamente, por exemplo, de alguns elementos de expressões tais como:

- (A) INSERT, REVOKE e SELECT.  
(B) CREATE, REVOKE e INSERT.  
(C) CREATE, GRANT e ALTER.  
(D) DROP, ALTER e UPDATE.  
(E) INSERT, INDEX e CREATE.

**Comentário:** Ok! Questão introdutória do assunto de banco de dados, relaciona os comandos da linguagem SQL a uma taxonomia específica. Na questão são apresentados o conjunto de comandos de criação de objetos ou *data definition language* (CREATE, DROP, ALTER), comandos relacionados à segurança dos dados ou *data control language* (GRANT e REVOKE) e os comandos utilizados para manipulação de dados ou *data manipulation language* (DELETE, INSERT, SELECT e UPDATE). Os comandos entre parênteses são apenas alguns exemplos dos representantes de cada subgrupo da linguagem SQL.

De posso dessas informações podemos encontrar nossa resposta na alternativa B, o que está de acordo com o gabarito apresentado pela banca.

**Gabarito: B.**

## 1.1. Linguagem procedural

A natureza envolvente do padrão SQL fez aumentar a quantidade de dialetos SQL disponível entre os vários fornecedores de plataformas. Estes dialetos comumente evoluem, pois, um determinado banco de dados de uma comunidade de usuários ou de um fornecedor requer habilidades na utilização do banco de dados antes do comitê da ANSI criar uma nova versão do padrão que aplique tal funcionalidade.

Ocasionalmente, porém, as comunidades acadêmicas e de pesquisa introduzem novas características em resposta a pressões de tecnologias concorrentes. Por exemplo, muitos fornecedores de banco de dados estão aumentando suas ofertas as funções programáticas tanto com Java (DB2, Oracle e Sybase) quanto com VBScript (Microsoft). Hoje em dia, programadores e desenvolvedores usam estas linguagens de programação em conjunto com SQL para construir programas.

Muitos destes dialetos incluem habilidades de processamento condicional (controladas por instruções do tipo IF ...THEN), funções de controle de fluxo (como laços WHILE e FOR), variáveis e habilidades de manipulação de erros. A ANSI não desenvolveu um padrão para estes atributos importantes quando os usuários começam a exigir, sendo assim desenvolvedores e fornecedores de SGBDR criaram seus próprios comandos e sintaxe.

É verdade que, alguns dos fornecedores mais antigos dos anos 80 tem variâncias nos comandos mais elementares já padronizados pela ANSI, como SELECT, pois suas implementações antecedem os padrões.

Alguns destes dialetos **introduziram comandos procedurais** para suportar a funcionalidade de uma linguagem de programação mais completa. Por exemplo, estas implementações procedurais contêm comandos para lidar com erros, linguagem de controle de fluxo, comandos condicionais, comandos para manipular variáveis, suporte para arrays e muitas outras extensões. Embora estas sejam implementações procedurais tecnicamente divergentes, são chamadas dialetos aqui. O pacote SQL/PSM (Persistent Stored Module) fornece muitos atributos associados com procedimentos armazenados e incorpora muitas das extensões oferecidas por estes dialetos. Vejam abaixo uma lista com os principais dialetos disponíveis no mercado.

**PL/SQL** - Encontrado em SGBDs Oracle. PL/SQL, que é a sigla para Procedural Language/SQL contém muitas semelhanças com a linguagem de programação geral. Esse conteúdo como falamos será visto em detalhes na nossa próxima aula.

**Transact-SQL** - Usado pelo Microsoft SQL Server e Sybase Adaptive Server. Como Microsoft e Sybase se afastaram, a plataforma comum que compartilhavam no início na década de 1990 foi dividida, suas implementações de agora também divergem, produzindo dois dialetos distintos de Transact-SQL.

**SQL PL** - extensão processual do IBM DB2 para SQL, introduzido na versão 7.0 do SGBD, fornece construções necessárias para a execução de controle de fluxo em torno de consultas SQL tradicionais e operações.

**PL/pgSQL** - O nome do dialeto SQL e das extensões implementadas no PostgreSQL. A sigla significa Procedural Language/PostgreSQL.

**MySQL** - O MySQL introduziu uma linguagem procedural em seu banco de dados na versão 5, mas não há nenhum nome oficial para ela. É concebível que como a Oracle adquiriu MySQL pode introduzir PL/SQL como parte do MySQL.

Aprender SQL vai proporcionar as habilidades que você precisa para recuperar informações a partir de qualquer banco de dados relacional. Também ajuda a compreender os mecanismos por trás das interfaces gráficas de consulta encontradas em muitos produtos de SGBDR. O entendimento de SQL ajuda você a criar consultas complexas e fornece o conhecimento necessário para corrigir consultas quando ocorrem problemas.

Vamos a seguir tratar detalhadamente dos comandos DDL e DML.

## 1.2. DDL

Quando estamos construindo um modelo de dados precisamos ter em mente a qual contexto ele está associado. Um modelo geralmente está associado a um esquema que faz parte de uma instância de um SGBD. Essa instância deve conter uma coleção de esquemas que recebe o nome de catálogo. Dentro destes esquemas temos um esquema especial conhecido como **INFORMATION SCHEMA**. Ele proporciona informações sobre todos os esquemas do catálogo e todos os descritores de seus elementos.

Este esquema contém definições para uma série de objetos de esquema, a maioria visões. Uma visão é uma tabela virtual que permite visualizar os dados coletados a partir de tabelas reais. Ao usar essas visões, você pode exibir as definições de objetos do catálogo como se fossem dados em SQL. Você não pode (deve) alterar qualquer um dos dados nestas visões - se você fizesse você estaria alterando as definições de objetos de si - mas você pode exibir informações simplesmente consultando a visão apropriada.

Os demais esquemas estão relacionados aos seus modelos de dados e podem ser vistos como um modo de agrupar as tabelas e outros construtores que pertencem à mesma aplicação de um banco de dados. Ele é identificado por um nome de esquema e inclui uma identificação de autorização, que indica o

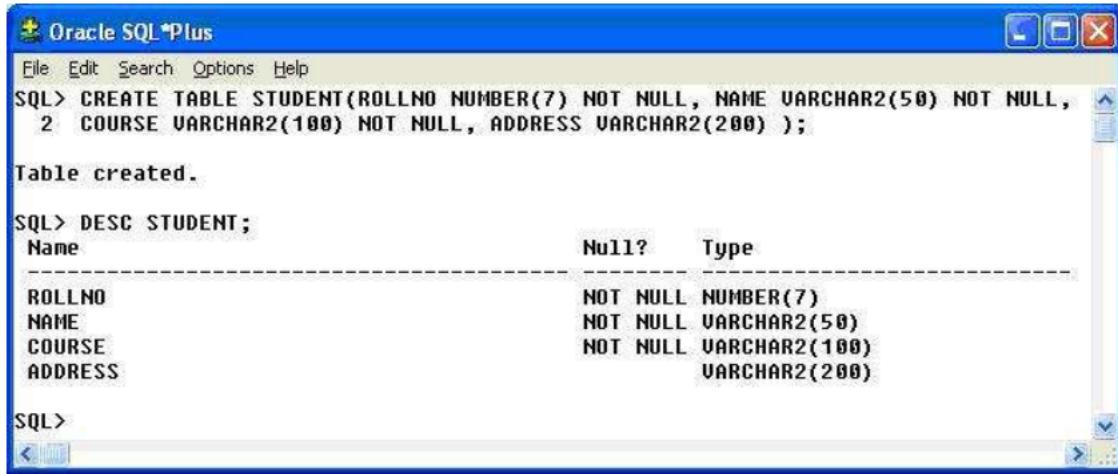
usuário ou a conta a qual o esquema pertence. Possui diversos elementos, tais como: tabelas, restrições, visões, domínios e outros construtores (como concessão de autoridade).

Para criar um SCHEMA é preciso que o privilégio para criação de esquemas seja explicitamente concedido. Geralmente apenas usuários relevantes, administradores de sistemas ou DBAs podem criar esquemas. Depois de criar um esquema é possível removê-lo. Deve-se utilizar o comando DROP SCHEMA, seguido pelo nome do esquema a ser excluído. Podendo ainda utilizar as opções RESTRICT e CASCADE após o nome do esquema.

Se a opção **CASCADE** for especificada, todos os objetos de esquema e de dados SQL dentro desses objetos são excluídos do sistema. Se a opção **RESTRICT** é usada, o esquema será excluído somente se não existir objetos de esquema.

### 1.2.1. Trabalhando com tabelas

Vamos agora conversar um pouco sobre tabelas, são várias as questões da que cobram o conteúdo deste assunto. A tabela é o local onde todos os dados são armazenados. Para criar uma tabela basta seguir a sintaxe do CREATE TABLE apresentada abaixo. Basicamente, você deve descrever as colunas que vão compor sua tabela, juntamente com os tipos e as restrições de integridade associados.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE STUDENT(ROLLNO NUMBER(7) NOT NULL, NAME VARCHAR2(50) NOT NULL,
2 COURSE VARCHAR2(100) NOT NULL, ADDRESS VARCHAR2(200) );
Table created.

SQL> DESC STUDENT;
Name          Null?    Type
ROLLNO        NOT NULL NUMBER(7)
NAME          NOT NULL VARCHAR2(50)
COURSE        NOT NULL VARCHAR2(100)
ADDRESS        VARCHAR2(200)

SQL>

```

Observem que optamos por dar um exemplo do Oracle. Aproveitamos para apresentar o comando DESC, que, seguido pelo nome da tabela, fornece uma descrição das colunas com seu tipo. O comando também informa se a coluna aceita ou não valores nulos.

Sobre o comando CREATE, ele é usado para especificar uma nova relação, dando-lhe um **nome** e especificando seus **atributos** e **restrições iniciais**. Vejam que os **atributos** são definidos **primeiro** e, a cada atributo, é dado **um nome, um tipo** para especificar o **domínio** de seus valores e alguma **restrição**

**de atributo.** As restrições de **chave**, de **integridade** de entidade e de integridade referencial **podem** ser específicas no mesmo comando após os atributos serem declarados ou poderão ser adicionadas depois usando o comando ALTER TABLE.

Outro comando importante é o comando DROP <TABLE> que remove todas as informações de uma relação do banco de dados. Em outras palavras exclui o objeto e os dados armazenados. É possível usar o modificador CASCADE para deletar também tabelas que sejam filhas da tabela removida ou pelo menos ajustar a integridade referencial.

Na alteração de tabelas feita pelo comando ALTER TABLE podemos adicionar e excluir atributos de uma relação. É possível definir um valor default para os novos atributos ou eles vão receber valores nulos para as tuplas já existentes. Vejam abaixo um resumo da sintaxe do alter table.

```
ALTER TABLE <table name>
    ADD [COLUMN] <column definition>
    | ALTER [COLUMN] <column name>
        { SET DEFAULT <default value> | DROP DEFAULT }
    | DROP [COLUMN] <column name> { CASCADE | RESTRICT }
```

Para entendermos melhor como escolher os tipos de dados das nossas colunas passaremos a seguir a analisar os principais tipos de dados de SQL e suas peculiaridades.

### 1.2.2. Tipos de dados

SQL inclui sete tipos de tipos de dados pré-definidos: string, numéricos, binário, datetime, interval, boolean e XML. Primeiramente vamos falar dos tipos de dados formados por cadeias de caracteres.

#### 1.2.2.1. CHARACTER

No SQL padrão esses tipos são divididos em tamanho fixo e variável. Todas as cadeias de caracteres em SQL podem ser de comprimento fixo ou variável. Você tem três tipos principais de cadeias de caracteres: 1. Cadeias de caracteres de tamanho fixo (CHARACTER ou CHAR), 2. Cadeias de caracteres de tamanho variável (CHARACTER VARYING ou VARCHAR) e 3. Cadeia de caracteres de para armazenar grandes objetos (CHARACTER LARGE OBJECT ou CLOB).

Como você pode imaginar, essa flexibilidade tem um preço de desempenho, pois o SGBD deve executar a tarefa adicional de alocação dinâmica. Um CHAR ou CHARACTER especifica o número exato de caracteres

que serão armazenados para cada valor. Por exemplo, se você definir o comprimento de 10 caracteres, mas o valor contém apenas seis caracteres os quatro caracteres restantes serão completados com espaços em branco. Um exemplo da sintaxe do comando seria: CONCURSO\_NOME CHAR (100).

O outro tipo de cadeia de caracteres é o CHARACTER VARYING ou VARYING CHAR ou VARCHAR. Ele especifica o número máximo de caracteres que pode ser incluído em uma variável. O número de caracteres armazenados é exatamente o mesmo número do valor introduzido, de modo que nenhum espaço será adicionado ao valor. Um exemplo da definição de uma coluna deste tipo: CONCURSO\_NOME VARCHAR (60). Observem que o comprimento da cadeia é passado entre parênteses.

O tipo de dados CHARACTER LARGE OBJECT (CLOB) foi introduzido juntamente com o SQL:1999. Como o próprio nome sugere, ele é usado para armazenar grandes cadeias de caracteres que são demasiadamente grandes para o tipo CHARACTER. CLOBs se comportam como cadeias de caracteres comuns, mas há uma série de restrições sobre o que você pode fazer com eles.

Por exemplo, um CLOB **não** pode ser usado em uma **chave primária**, **chave estrangeira** ou com predicado **UNIQUE**. Devido ao seu tamanho grande, aplicações geralmente não transferem o CLOB para ou a partir de um banco de dados. Em vez disso, um tipo de dados especial do lado do cliente chamado CLOB *locator* é utilizado para manipular os dados CLOB. É um parâmetro cujo valor identifica um objeto grande.

Vários idiomas têm alguns CHARACTER que diferem de quaisquer caracteres em outro idioma. Por exemplo, o alemão tem alguns caracteres especiais não presentes no conjunto de caracteres do idioma Inglês. Você pode especificar o idioma Inglês definindo-o como o padrão para o seu sistema, mas você pode usar outros conjuntos de caracteres alternativos para isso existe o NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, e NATIONAL CHARACTER LARGE OBJECT. Eles são tipos de dados com a mesma função que o CHARACTER, CHARACTER VARYING, e CHARACTER LARGE OBJECT - a única diferença é que o conjunto de caracteres que você está especificando é diferente do conjunto de caracteres padrão.

Você pode especificar o conjunto de caracteres quando você define uma coluna de uma tabela. Se você quiser, cada coluna pode usar um conjunto de caracteres diferente. O exemplo a seguir mostra a criação de uma tabela e usa vários conjuntos de caracteres:

```
CREATE TABLE IDIOMAS (
    LANGUAGE_1 CHARACTER (40),
    LANGUAGE_2 CHARACTER VARYING (40) CHARACTER SET GREEK,
    LANGUAGE_3 NATIONAL CHARACTER (40),
    LANGUAGE_4 CHARACTER (40) CHARACTER SET KANJI
```

) ;

Aqui a coluna LANGUAGE\_1 contém caracteres em conjunto de caracteres padrão do sistema. A coluna LANGUAGE\_3 contém caracteres em conjunto de caracteres nacional do sistema. A coluna LANGUAGE\_2 contém caracteres gregos. E a coluna LANGUAGE\_4 contém caracteres Kanji. Depois de uma longa ausência, conjuntos de caracteres asiáticos, como Kanji, estão agora disponíveis em muitos produtos.

### 1.2.2.2. Binary String

Ainda dentro das strings temos as strings binárias. Uma string binária é uma sequência de bytes da mesma forma que uma string de caracteres é uma sequência de caracteres. Ao contrário das strings de caracteres que geralmente contêm informações na forma de texto, ela é usada para armazenar dados não tradicionais, tais como imagens, áudio e arquivos de vídeo, executáveis de programa, e assim por diante.

Os tipos de dados de strings binárias foram introduzidos no SQL:2008. Considerando que os dados binários têm sido fundamentais para computadores digitais desde o Atanasoff-Berry Computer (primeiro computador eletrônico digital) da década de 1930, esse reconhecimento da importância dos dados binários parece um pouco tarde em vir para SQL. Antes tarde do que nunca! Existem três tipos diferentes BINARY, BINARY VARYING, e BINARY LARGE OBJECT. BINARY LARGE OBJECT é conhecido como BLOB. O BLOB armazena grandes grupos de bytes, até o valor especificado.

Estes são os principais tipos de dados de caracteres. Apresentamos abaixo uma lista que compara os tipos de caracteres definidos pelo SQL padrão com os SGBDs comercial.

SQL STANDARD	ORACLE 11G	DB2 9.7	MS SQL SERVER 2008
CHARACTER	CHARACTER	CHARACTER	CHARACTER
VARYINGVARCHAR	VARYING VARCHAR VARCHAR2 LONG VARCHAR	VARYING VARCHAR LONG VARCHAR	VARYING VARCHAR TEXT
CLOB or CHARACTER LARGE OBJECT	CLOB	CLOB	VARCHAR (MAX)
NCHAR	NCHAR	GRAPHIC	NCHAR
NCHAR VARYING(n)	NCHAR VARYING NVARCHAR2	VARGRAPHIC LONG VARGRAPHIC	NVARCHAR
NATIONALCHARACTER LARGE OBJECT	NCLOB	DBCLOB	NVARCHAR (MAX)

### 1.2.2.3. Tipos numéricos

Vamos agora tratar dos tipos de dados numéricos. Eles são divididos em duas grandes categorias: números exatos (*exact numbers*) e números aproximados (*approximate*).

Os números exatos podem ser números inteiros (lápis, pessoas ou planetas) ou ter casas decimais (preços, pesos ou percentuais). Os números podem ser positivos e negativos. Eles podem ter precisão e escala. O que seria isso? A precisão (p) determina o número total máximo de dígitos decimais que podem ser armazenados (tanto a esquerda quanto a direita do ponto decimal). E escala (e) especifica o número máximo de casas decimais permitidas. Eles estão disponíveis da seguinte forma, por exemplo, no SGBD Oracle.

SQL STANDARD	ORACLE 11G
INTEGER	NUMBER (38) INT
SMALLINT	SMALLINT NUMBER (38)
NUMERIC	NUMERIC DECIMAL NUMBER

Os números de ponto flutuantes ou aproximados são números que não podem ser representados com precisão absoluta. Vejam a abaixo a relação entre os tipos SQL padrão e o ORACLE.

SQL STANDARD	ORACLE 11G
FLOAT	FLOAT NUMBER
REAL	REAL NUMBER
DOUBLE PRECISION	DOUBLE PRECISION NUMBER

Para concluir o nosso estudo sobre tipos de dados numéricos apresentamos uma tabela que relaciona os tipos com o espaço de armazenamento necessário para cada valor associado e o range, que seria os valores possíveis ou domínio de um determinado tipo numérico.

DATA TYPE	STORAGE SIZE (BYTES)	RANGE
INTEGER	4	-2,147,483,648 to +2,147,483,647
TINYINT	1	0 through 255
SMALLINT	2	-32,768 to +32,768
BIGINT	8	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,808
MONEY	8	-922,337,203,685,477.5808 to +922,337,203,685,477.5807
SMALLMONEY	4	-214,748.3648 to +214,748.3647
REAL	4	The range is from negative 3.402E + 38 to negative 1.175E - 37, or from positive 1.175E - 37 to 3.402E + 38. It also includes 0.
FLOAT	4 to 8	The number can be zero or can range from -1.79769E + 308 to -2.225E - 307, or from 2.225E - 307 to 1.79769E + 308.
DOUBLE	8	The number can be zero or can range from -1.79769E + 308 to -2.225E - 307, or from 2.225E - 307 to 1.79769E + 308.

#### 1.2.2.4. Datetime e interval

Vamos agora falar sobre o tipo DATE. DATE é uma estrutura que consiste em três elementos: ano, mês e dia. O ano é um número de quatro dígitos que permite que os valores de 0000 a 9999. O mês é um elemento de dois dígitos com valores de 01 a 12, e dia que tem dois dígitos com um intervalo de 01 a 31. SQL/ANSI define a semântica de data usando a estrutura descrita acima, mas os SGBDs não são obrigados a usar essa abordagem, desde a implementação produza os mesmos resultados.

O tipo TIME consiste de hora, minuto e segundo. A hora é um número de 00 a 23. O minuto é um número de dois algarismos, de 00 a 59. O segundo é um inteiro entre 00-59 ou um número decimal com uma escala mínima de cinco e precisão mínima de três que pode conter valores de 00.000 para 59.999.

Temos ainda os tipos: **1. DATETIME** que combina data e hora em um único tipo, com intervalo de datas. **2. TIMESTAMP** que engloba os campos DATE e TIME, mais seis posições para a fração decimal de segundos e uma

qualificação opcional WITH TIME ZONE e **3. INTERVAL** que serve para calcular o intervalo entre dois objetos que representam tempos.

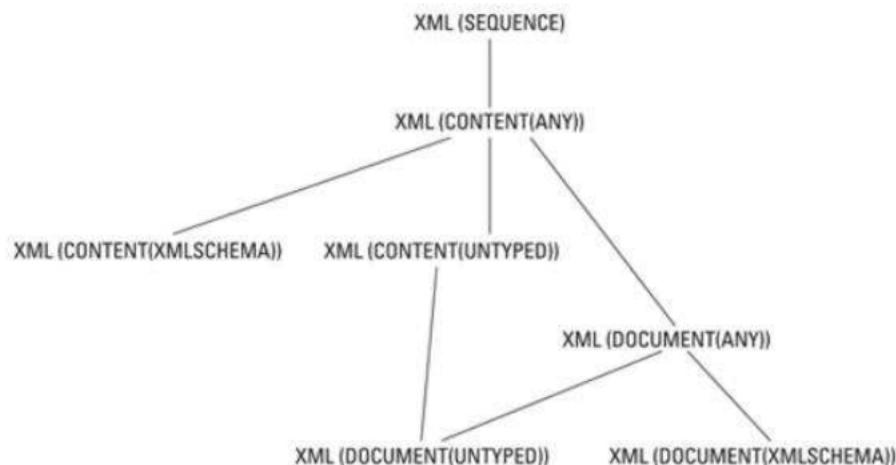
### 1.2.2.5. Boolean e XML

O tipo de dados BOOLEAN consiste na verdade dos valores distintos, verdadeiro e falso, assim como o valor desconhecido. O valor verdadeiro ou falso booleano pode comparado com um valor NULL.

XML é um acrônimo para eXtensible Markup Language, que define um conjunto de regras para a adição de marcação aos dados. As estruturas de marcação permitem aos dados transmitir o que eles significam. XML permite o compartilhamento de dados entre diferentes plataformas.

O tipo de dados XML tem uma estrutura de árvore, assim um nó raiz pode ter nós filho, o que pode, por sua vez, ter seus próprios filhos. Introduzido pela primeira vez no SQL:2003, o tipo XML foi concretizado na versão SQL/XML: 2005, e ainda mais melhorado no SQL:2008.

Os modificadores primários do tipo XML são SEQUENCE, CONTENT e DOCUMENT. Os modificadores secundários são UNTYPED, ANY e XMLSCHEMA. A figura mostra a estrutura de árvore que ilustra as relações hierárquicas entre os subtipos.



Por fim, é importante saber que o valor nulo é membro de todos os tipos de domínios. Se você declarar o domínio de um atributo como NOT NULL o SGBD vai proibir a inserção de valores nulos para essa coluna.

### 1.2.2.6. Tipos ROW, Collections e UDT

Além dos tipos predefinidos, embutidos, SQL suporta tipos de coleção, tipos construídos e tipos definidos pelo usuário (UDT).

#### Tipos ROW

O tipo de dados de linha foi introduzido com o SQL:1999. Uma coisa notável sobre o tipo de dados ROW é que ela viola as regras de normalização declaradas por Codd nos primeiros dias da teoria banco de dados relacional. Uma das características que definem a primeira forma normal é que um atributo de uma linha da tabela não pode ter um valor múltiplo. Um campo pode conter um e apenas um valor, ou seja, é atômico. No entanto, o tipo de dados ROW permite que você declare uma linha inteira de dados contida dentro de um único campo em uma única linha de uma tabela - em outras palavras, uma linha dentro de outra.

As formas normais, definidas por Codd, são características que definem a estrutura para bancos de dados relacionais. A inclusão do tipo ROW no padrão SQL foi a primeira tentativa de ampliar SQL além do modelo relacional puro. Considere a seguinte instrução SQL, que define um tipo ROW para informações sobre o endereço de uma pessoa:

```
CREATE ROW TYPE addr_typ (
    Street      CHARACTER VARYING (25),
    City        CHARACTER VARYING (20),
    State       CHARACTER (2),
    PostalCode  CHARACTER VARYING (9)
);
```

Depois que ele é definido, um novo tipo de linha pode ser usado em uma definição de tabela:

```
CREATE TABLE CUSTOMER (
    CustID     INTEGER   PRIMARY KEY,
    LastName   CHARACTER VARYING (25),
    FirstName  CHARACTER VARYING (20),
    Address    addr_typ,
    Phone      CHARACTER VARYING (15)
);
```

A vantagem aqui é que se você está mantendo informações de endereço para diferentes entidades - como clientes, fornecedores, colaboradores e acionistas - você tem que definir os detalhes da especificação de endereço apenas uma vez: na definição do tipo ROW.

### Tipos de coleção

Após SQL sair da fronteira do modelo relacional com o SQL:1999, os tipos de dados que violam a primeira forma normal tornaram-se possíveis. Um campo pode conter uma coleção de objetos, em vez de apenas um. O tipo ARRAY foi introduzido no SQL:1999, e o tipo MULTISSET foi introduzido no SQL:2003.

Duas coleções podem ser comparadas entre si somente se ambas são do mesmo tipo, ou ARRAY ou MULTISSET, e se os seus tipos de elementos são

compatíveis. Como ARRAYS têm uma ordem para os elementos definidos, elementos correspondentes podem ser comparados. MULTISETS não tem nenhuma ordem definida para os elementos, mas você pode compará-los, se (a) existe uma enumeração sobre cada um deles e (b) as enumerações podem ser emparelhadas.

### Tipos definidos pelo usuário (UDT – User defined types)

Tipos definidos pelo usuário (UDTs) representam outro exemplo de recursos que chegaram no SQL:1999 que vêm do mundo de programação orientada a objetos. Como um programador de SQL, você não está mais restrito aos tipos de dados definidos na especificação SQL. Você pode definir seus próprios tipos de dados, usando os princípios de tipos de dados abstratos (ADTs) encontrados em tais linguagens de programação orientada a objetos como C++.

Um dos benefícios mais importantes dos UDTs é o fato de que você pode usá-los para eliminar a diferença de impedância entre o SQL e a linguagem de host que está "envolvendo" o SQL. Um problema de longa data com o SQL tem sido o fato de tipos de dados predefinidos do SQL não correspondem aos tipos de dados das linguagens host no qual as instruções SQL são incorporadas. Agora, com UDTs, um programador de banco de dados pode criar tipos de dados dentro do SQL que correspondem aos tipos de dados da linguagem de host.

UDT tem atributos e métodos, que são encapsulados. O mundo exterior pode ver as definições de atributos e os resultados obtidos pelos métodos - mas as implementações específicas dos métodos estão escondidas. O acesso aos atributos e métodos de uma UDT pode ser ainda mais restrito, podemos especificá-los como públicos, privados ou protegidos.

Agora que vimos os tipos de dados vamos passar rapidamente pelas possíveis restrições de integridade que compõe o padrão SQL e vão fazer parte dos comandos de criação de tabelas.

#### 1.1.1. Restrições de integridade

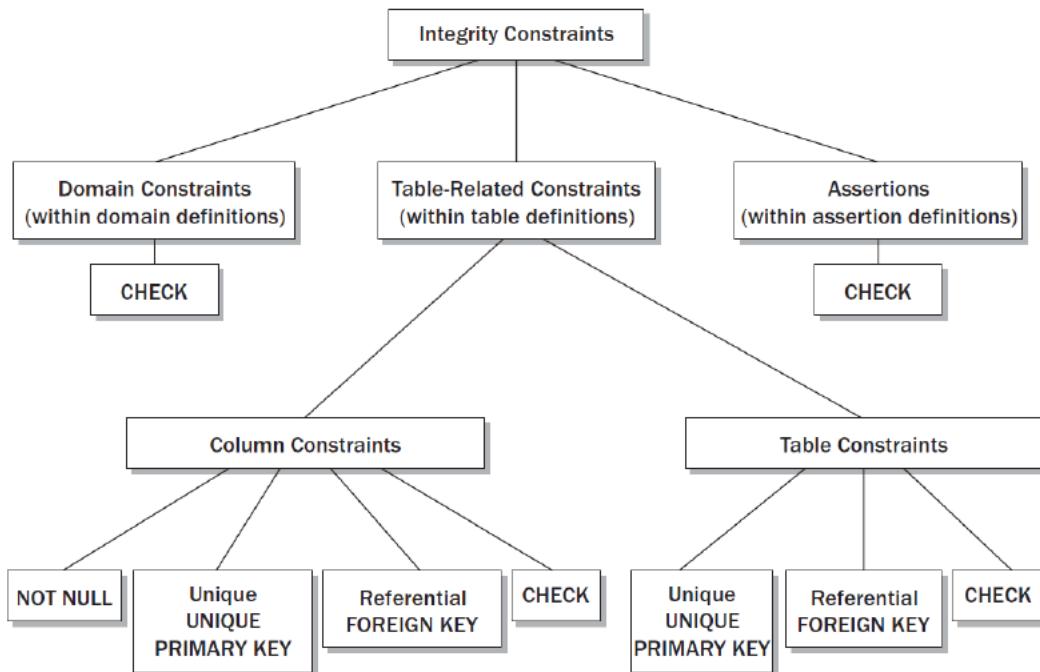
Restrições de integridade SQL, que são geralmente referidas como restrições, podem ser divididos em três categorias:

- **Restrições em tabelas** – Definida dentro de uma definição de tabela. A restrição pode ser definida como parte da definição de coluna ou como um elemento ao final da definição. Restrições definidas no nível de tabela podem ser aplicadas a uma ou mais colunas.

- **Assertions (Afirmações)** - Um tipo de restrição que é definida dentro de uma definição afirmação (separado da definição da tabela). Uma afirmação pode ser relacionada com **uma ou mais tabelas**.

- **Restrições de domínio** - Um tipo de restrição que é definida dentro de uma definição de domínio (separado da definição da tabela). A restrição de domínio **está associada com qualquer coluna** que é definida baseada no domínio específico. Aqui você pode criar um novo domínio para atribuir o mesmo a uma coluna de uma tabela.

Para tentar clarear um pouco mais observe a figura abaixo que apresenta o comando necessário para “ativar” cada uma das restrições e a distribuição e uso destes comandos de acordo com a classificação acima.



Falaremos um pouco sobre cada uma delas. A primeira seria a restrição de chave primária que pode ser simples, neste caso basta colocar as palavras PRIMARY KEY ao lado da definição da coluna na construção do CREATE TABLE, ou composta adicionando ao final das definições das colunas PRIMARY KEY (col1, col2, ..., coln). Vejam que entre parênteses aparece o nome das colunas que fazem parte da chave, separadas por vírgulas.

A próxima restrição seria a restrição de integridade referencial. Esta restrição trata do relacionamento entre tabelas. Quando temos um atributo da tabela A sendo referenciado como atributo de uma tabela B temos que garantir que os valores presentes em B sejam válidos. Esses valores têm, portanto, que estar presentes ou armazenados dentro da coluna referenciada ou serem nulos.

Ao criarmos uma integridade referencial podemos incluir nela uma ação referencial engatilhada ou REFERENCIAL TRIGGERED ACTION. Ela basicamente vai optar por executar uma ação (CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION) quando for executado uma operação de DELETE ou UPDATE em uma tabela. Veja os exemplos abaixo.



```
CONSTRAINT EMPSUPERFK
    FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO (SSN)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
CONSTRAINT EMPDEPTFK
    FOREIGN KEY (DNO) REFERENCES DEPARTAMENTO (DNUMERO)
        ON DELETE SET DEFAULT ON UPDATE CASCADE );
```

É possível ainda termos as seguintes restrições: 1. NOT NULL que especifica que uma coluna não pode receber valores nulos; 2. UNIQUE que define que uma coluna não pode ter valores duplicados; 3. DEFAULT que atribui um valor padrão para uma determinada coluna a ser utilizado, caso nenhum valor seja passado como parâmetro no momento da inserção; 4. CHECK utilizado para restringir os valores de domínio, verificando se eles estão contidos no conjunto de valores especificados.

O CHECK é a restrição mais flexível de todas. Sua sintaxe básica é relativamente simples. Para criar uma restrição do tipo CHECK em uma coluna você deve utilizar a seguinte sintaxe: <column name> { <data type> | <domain> } CHECK ( <search condition> ). No caso da criação da restrição em uma tabela use: [CONSTRAINT <constraint name> ] CHECK ( <search condition> ).

Agora que já aprendemos as restrições de integridade já temos a capacidade de consolidar nosso conhecimento em DDL fazendo a questão abaixo.

**03. BANCA: FCC ANO: 2015 ÓRGÃO: TRT - 15ª REGIÃO (CAMPINAS-SP)  
PROVA: TÉCNICO JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO**

Deseja-se criar uma tabela chamada Departamento contendo os seguintes campos:

idDep – inteiro, chave primária, não nulo, auto numeração.

nomeDep – cadeia de caracteres com, no máximo, 50 caracteres, não nulo.

telefoneDep – cadeia de caracteres com, no máximo, 15 caracteres.

Considerando-se que o banco de dados está aberto e em condições ideais, o comando SQL que deverá ser utilizado é:

A CREATE TABLE Departamento ( idDep INT NOT NULL AUTO\_INCREMENT PRIMARY\_KEY, nomeDep VARCHAR(50) NOT NULL, telefoneDep VARCHAR(15));

B CREATE TABLE Departamento ( idDep INT NOT NULL AUTO\_NUMBERING, nomeDep VARCHAR(50) NOT NULL, telefoneDep VARCHAR(15), PRIMARY KEY (idDep));

C CREATE TABLE Departamento ( idDep INT NOT NULL AUTOINCREMENT, nomeDep VARCHAR2(50) NOT NULL, telefoneDep VARCHAR2(15) NULLABLE, PRIMARY KEY (idDep));

D CREATE TABLE Departamento ( idDep INT NOT NULL AUTO\_INCREMENT, nomeDep VARCHAR(50) NOT NULL, telefoneDep VARCHAR(15), PRIMARY KEY (idDep));

E CREATE TABLE Departamento ( idDep INT NOT NULL AUTOINCREMENT, nomeDep VARCHAR(50) NOT NULL, telefoneDep VARCHAR(15) PRIMARY\_KEY (idDep));

**Comentário:** Para respondermos a essa questão vamos precisar de uma informação a mais não tratada até o momento. É possível em SQL criar colunas que seja auto incrementadas e para isso basta colocarmos o modificador AUTO\_INCREMENT ao lado da definição da coluna. Agora podemos tentar montar o comando CREATE solicitado pela questão. Primeiramente começamos com o "CREATE TABLE Departamento ("

Sabemos que a primeira variável é idDep é do tipo inteiro (INT), não nulo (NOT NULL), auto numerado (AUTO\_INCREMENT) e chave primária (PRIMARY KEY). A coluna nomeDep também é não nula (NOT NULL), possui no máximo 50 caracteres, vejam que temos que usar um tipo de caracteres variável (VARCHAR(50)). Por fim a coluna telefoneDep, que é do tipo cadeia de caracteres com no máximo 15 caracteres (VARCHAR(15)).

Observe que a definição da chave primária pode vir junto com a coluna ou ao final da definição da coluna por meio da palavra chave PRIMARY KEY(idDep). Esta foi a forma utilizada pela questão. Se montarmos o quebra cabeça com os termos acima temos exatamente o conteúdo descrito na alternativa D, que é a nossa resposta.

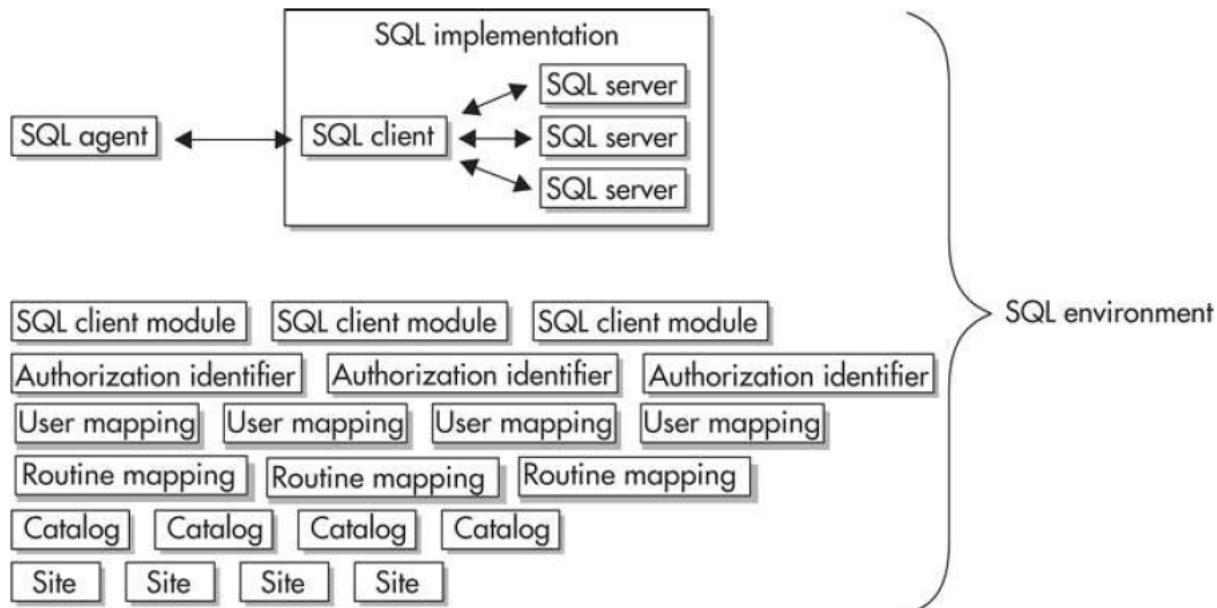
**Gabarito: D.**

## 1.2. Entendendo o catálogo SQL - (INFORMATION\_SCHEMA)

Quando falamos da estrutura ou do ambiente de SQL devemos ter a noção que ele é descrito como a soma das partes que fazem parte como que o ambiente funcione de forma correta. Cada uma dessas partes, ou componentes, atuam em conjunto com outros componentes para dar suporte às operações como a criação e modificação de objetos, armazenamento e consulta de dados, ou ainda modificação e deleção destes dados.

A união desses componentes compõe um modelo no qual os SGBDs se baseiam. Isso não significa que um SGBD seguirá estritamente essa orientação

quando implementa suas estruturas. A forma pela qual eles implementam e quais os componentes são implementados é uma decisão de cada equipe de desenvolvimento. Mesmo assim, o padrão SQL ANSI define e descreve alguns componentes distintos. São sei tipos de componentes: O cliente SQL e o servidor SQL são parte da implementação SQL. Vejam a figura abaixo:



O **SQL agent** é qualquer estrutura que leva a execução um comando SQL. O agente SQL é ligado a um *SQL client* por meio da implementação SQL,

A **SQL implementation** é o processo que executa comandos SQL de acordo com as requisições de um agente SQL. A implementação inclui um cliente SQL e um ou mais servidores SQL. O cliente estabelece as conexões SQL como os servidores e mantem os dados relacionados com a interação entre o SQL agent e os SQL servers.

O **SQL client module** ajuda na chamada de uma coleção de comandos SQL que são escritos separadamente do código da aplicação do seu programa. Neste contexto imagine que você tem alguns códigos SQL que são invocados dentro do seu código JAVA. Esses códigos são comandos SQL simples.

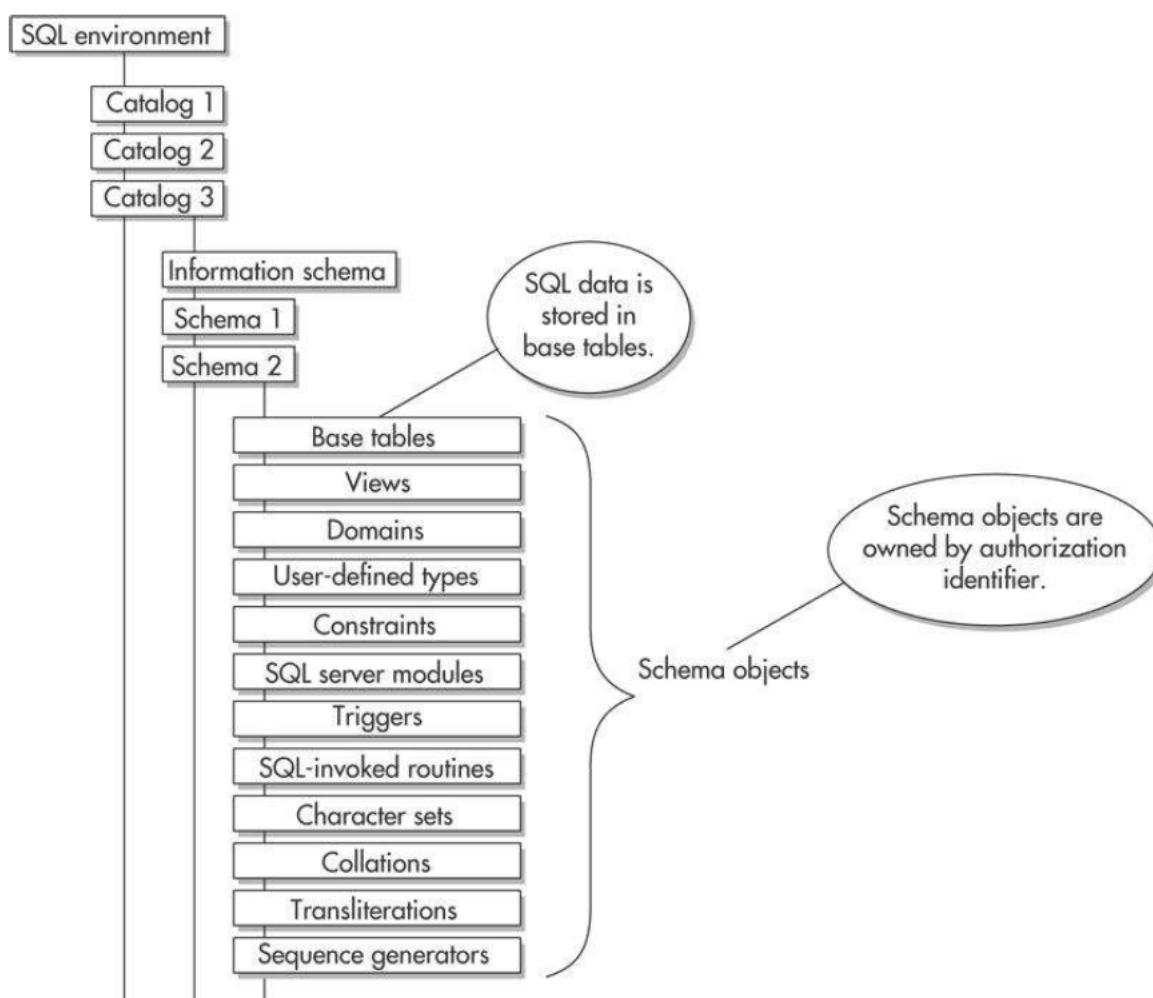
O **identificador de autorização** é uma estrutura que permite identificar quais os privilégios são atribuídos a um usuário ou uma role e seu relacionamento com os objetos e dados. Um usuário é uma conta de segurança individual que representa um indivíduo, uma aplicação ou um serviço de sistema. Um papel ou role é um conjunto predefinido de privilégios que pode ser designado para um usuário ou a outra role.

O **mapeamento de usuário** (*user mapping*) relaciona um identificador de autorização com um descritor de servidor externo. O **mapeamento de rotina** (*routine mapping*) relaciona uma rotina SQL invocada a um descritor de servidor externo.

Um **catálogo** é um grupo de esquemas agrupados sobre um determinado nome. Cada catálogo contém a informações sobre os esquemas, que incluem uma descrição sobre os objetos dos esquemas. O catálogo prover uma estrutura hierárquica para a organização dos dados dentro dos esquemas. Um esquema é um contêiner para objetos como tabelas, visões e domínios.

Um **site** é um grupo de tabelas que contém dados de SQL, é descrito como o conteúdo dos esquemas. Os dados são conhecidos pelo nome de banco de dados.

Outra forma de observarmos um catálogo é como uma estrutura hierárquica, onde o catálogo é o objeto pai de um conjunto de outros objetos esquema. No topo da hierarquia temos o ambiente SQL, que contém zero ou mais catálogos. Os esquemas são localizados numa terceira camada abaixo do ambiente e do catálogo, e os objetos de esquema são localizados em uma quarta camada.



### 1.2.1. Information Schema

Cada catálogo contém um esquema especial conhecido como INFORMATION\_SCHEMA. Esse esquema contém um conjunto de objetos de esquemas, geralmente views. Um view é uma tabela virtual que permite ver dados coletados a partir das tabelas atuais. Usando visões, você consegue exibir as definições dos objetos presentes no catálogo como se fossem dados armazenados. Contudo, você não pode modificar qualquer um destes dados, se você fizer estará modificando a definição do objeto em si, você pode apenas exibir as informações por meio de consultas a visão apropriada.

Tal como acontece com a maioria dos recursos de SQL, a implementação do INFORMATION\_SCHEMA e quais funções são suportadas variam de produto para produto, embora essas implementações são geralmente bastante simples. Por exemplo, o SQL Server 2012 inclui uma visão no esquema de informações chamada INFORMATION\_SCHEMA.COLUMNS. Se você consultar esta view, os resultados incluirão uma lista que contém informações sobre cada coluna acessível para o usuário atual do banco de dados. Os resultados incluem informações como o nome da coluna, o tipo de dados atribuído a essa coluna, e o proprietário (identificador de autorização) que possui essa coluna.

Agora que apresentamos os conceitos relacionados a ambiente e catálogo SQL, podemos seguir nosso estudo com o grupo de comandos de manipulação de dados.

## 1.3. DML

São quatro os principais comandos DML: SELECT, INSERT, UPDATE e DELETE. Todos os SGBDs relacionais implementam esses comandos. Vamos conhecer a sintaxe de cada um deles. É importante perceber que a maioria das peculiaridades estão descritas para o comando SELECT.

### 1.3.1. O COMANDO SELECT

A instrução SELECT permite formar consultas complexas que podem retornar exatamente o tipo de dados que você deseja recuperar. Como você pode observar abaixo, as cláusulas exigidas na construção de um comando SELECT são a palavra reservada SELECT e a cláusula FROM. Todas as outras cláusulas são opcionais! É importante relembrarmos que este comando possui uma correspondência com a álgebra relacional. Assim a projeção corresponde às colunas definidas no SELECT, e a seleção da álgebra relacional corresponde às condições de busca inseridas na cláusula WHERE. Podemos ainda observar um produto cartesiano entre as tabelas na cláusula FROM.



```
SELECT [ DISTINCT | ALL ] { * | <select list> }
FROM <table reference> [ { , <table reference> } . . . ]
[ WHERE <search condition> ]
[ GROUP BY <grouping specification> ]
[ HAVING <search condition> ]
[ ORDER BY <order condition> ]
```

Sobre o comando acima a primeira informação importante é que o \* (asterisco) é utilizado quando queremos extrair na nossa consulta todas as colunas da tabela, ou das relações descritas na cláusula FROM. Quando nossa intenção for recuperar também todas as linhas devemos omitir a cláusula WHERE. Nela são informadas as restrições que queremos fazer sobre a nossa busca. Podemos, por exemplo, restringir o domínio de um valor inteiro selecionando apenas as tuplas que satisfazem a essa restrição.

Quando o SGBD recebe uma consulta com todas as palavras chaves acima em segue uma ordem lógica para avaliação do comando. Primeiramente ele analisa as tabelas ou relações envolvidas na consulta que estão presentes no FROM. Em seguida, o SGBD vai verificar as restrições de busca ou condições contidas na cláusula WHERE. Dando prosseguimento, caso exista uma especificação de agrupamento descrita no GROUP BY ela é verificada.

Essa condição de agrupamento geralmente vem acompanhada de uma função de agregação sobre uma determinada coluna da tabela. Podemos pensar, por exemplo, na operação de soma (SUM) dos valores de um atributo numérico da tabela. Após o agrupamento é possível fazer uma restrição sobre os valores agrupados. Pela descrição da norma SQL/ANSI você pode comparar o resultado dos agrupamentos e selecionar apenas aqueles que satisfaçam a uma condição de busca.

Após essa etapa o SGBD vai avaliar quais colunas são descritas na cláusula SELECT do comando. Por fim, é possível ordenar a relação pelos atributos ou colunas listadas no ORDER BY.

No Oracle é possível definir uma variável prefixada utilizando o & para avisar ao banco de dados que o valor será informado no momento da execução do comando. Veja nas figuras abaixo o uso do & para uma variável numérica, e para o caso do valor recebido ser uma data ou uma string de caracteres onde deve ser usado aspas simples.

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num;
```

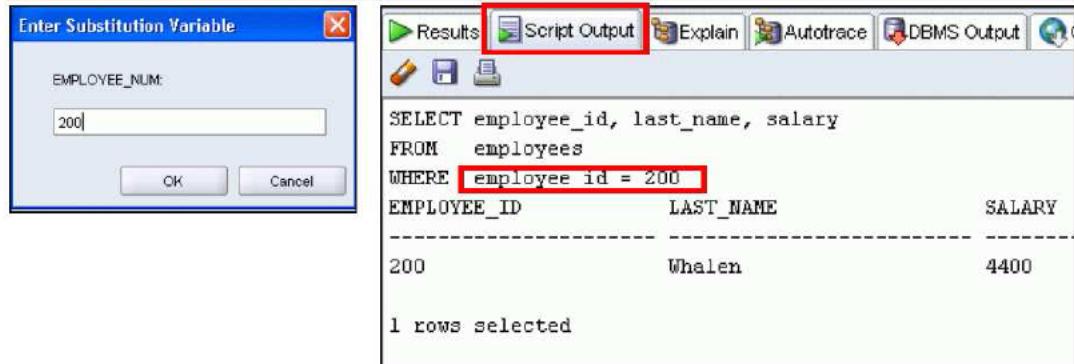
```
SELECT last_name, department_id, salary*12
FROM employees
WHERE job_id = '&job_title' ;
```

Outro ponto interessante no uso de variáveis é quando você quer reusar o valor da variável sem necessitar de digitar novamente ou repassar um novo valor. Neste caso você deve usar dois & 's comerciais. Veja a figura abaixo:

```
SELECT employee_id, last_name, job_id, &&column_name
FROM employees
ORDER BY &column_name ;
```

Use o comando VERIFY para alternar a exibição da variável de substituição, tanto antes como depois SQL Developer substitui variáveis de substituição com valores:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = &employee_num;
```



### 1.3.2. ALIAS, DISTINCT e operadores

A Linguagem SQL possui um mecanismo para renomear tanto as relações quanto os atributos através da cláusula **as**, da seguinte forma: nome\_antigo AS novo\_nome. Esse mecanismo é conhecido como **alias**. Quando utilizado no contexto de uma tabela na cláusula FROM os alias são conhecidos como variáveis de tuplas.

Existem diversos modificadores de consulta que nos ajudam a adequar o retorno da nossa consulta. Para forçar a eliminação de tuplas duplicadas

devemos inserir a declaração **DISTINCT** depois do SELECT. A cláusula SELECT pode conter expressões aritméticas envolvendo os operadores +, -, \*, /, em operações que utilizam constantes ou atributos de tabelas.

Não é necessário que os atributos usados na cláusula WHERE estejam listados no SELECT. A cláusula corresponde ao predicado de seleção da álgebra relacional como já falamos. Ela consiste de um predicado envolvendo atributos das relações que aparecem na cláusula FROM. Podemos utilizar operadores aritméticos (+, -, \*, /, % (módulo)), operadores de comparação (<, <=, >, >=, = e <>) e ainda conectivos ou operadores lógicos (AND, OR e NOT).

Outra possibilidade de comparação existente é quando estamos comparando STRINGS. Neste caso podemos utilizar os operadores LIKE, NOT LIKE, % (que combina qualquer substring, independente do tamanho) e \_ (combina caractere a caractere). Basicamente o que fazemos com esses operadores é verificar se um valor de um atributo em uma determinada tupla ele é semelhante ao de uma constante passada como parâmetro. Vejamos um exemplo, se você quiser encontrar o nome de todos os clientes cuja rua contenha a substring 'Professor' você pode executar o comando abaixo.

```
select nome_cliente
from cliente
where rua_cliente like '% Professor %';
```

Quando construímos um predicado com vários operadores precisamos entender que existe uma precedência na avaliação dos operadores. Você pode usar parênteses para sobrescrever as regras de precedência. A ordem de avaliação das regras de precedências pode ser vista na lista abaixo:



Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Depois de definir a cláusula WHERE podemos ordenar o resultado da consulta por meio do comando ORDER BY. É possível ordenar por um determinado atributo em ordem crescente e outro em ordem decrescente. Para isso basta seguir seguinte sintaxe: ORDER BY <nome(s)\_coluna(s)> DESC ou ASC (**default**) e separar por vírgula os nomes das colunas com suas respectivas formas de ordenação. Lembrando que se não for definida a ordenação padrão é feita de forma ascendente.

Para executarmos a ordenação pode optar por usar valores numéricos que referenciam a enésima coluna que aparece na cláusula SELECT. Veja o exemplo na figura abaixo, cujo valor **3** referencia a coluna *hire\_date*:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```

Vejamos um exemplo de questão de provas anteriores.



**2. Ano: 2016 Banca: FCC Órgão: TRE-SP Cargo: Analista Judiciário de TI – Discursiva. QUESTÃO 2**

Considere que, em uma situação hipotética, uma equipe de Analistas de Sistemas do TRE-SP irá participar do desenvolvimento de um novo sistema com base na metodologia ágil Scrum. Para a escolha do Time Scrum, algumas tarefas foram solicitadas para definição do Product Owner, do responsável pelo Backlog do Produto e dos membros da Equipe de Desenvolvimento. Como base do teste, foi criada uma tabela no banco de dados denominada ELEICOES, que tem os campos indicados na 1a linha e os conteúdos possíveis na 2a linha, conforme abaixo.

NomeCandidato	Cargo	VotosValidos	Sexo	CodigoPartido
Até 100 caracteres	Prefeito Vereador	Numérico inteiro positivo	Feminino Masculino	1 a 10

Neste contexto, solicita-se que sejam apresentadas soluções para o que se pede abaixo.

- Escrever um comando SQL para mostrar, em uma linha, a quantidade de candidatos do sexo masculino como "CandidatosHomens" e a quantidade de candidatas do sexo feminino como "CandidatasMulheres".
- Escrever um comando SQL para apresentar todos os dados dos candidatos a Prefeito por ordem crescente de VotosValidos.

**Comentário:** Vamos tentar apresentar uma solução para cada uma das alternativas:

- SELECT \* FROM  
(SELECT COUNT(SEXO) AS "CandidatosHomens"  
FROM ELEICOES WHERE SEXO="Masculino") as M,

```
(SELECT COUNT(SEXO) AS "CandidatosMulheres"
FROM ELEICOES WHERE SEXO="Feminino") as F;
b. SELECT * FROM ELEICOES WHERE CARGO = "Prefeito"
ORDER BY VotosValidos ASC;
```

**Gabarito: Discursiva!**

### 1.3.3. Consultas aninhadas (IN, EXISTS, ALL, SOME, ANY)

Algumas consultas precisam que os valores existentes no banco de dados sejam buscados e depois usados em uma condição de comparação. SQL provê um mecanismo para aninhamento de subconsultas. Uma subconsulta é uma expressão do tipo SELECT-FROM-WHERE que é aninhada dentro de outra consulta. As aplicações mais comuns para as subconsultas são testes para membros de conjuntos, comparação de conjuntos e cardinalidade de conjuntos.

Sempre que uma condição na cláusula WHERE de uma consulta aninhada referencia algum atributo de uma relação declarada na consulta externa, as duas consultas são consideradas correlacionadas. Vejamos um exemplo. Vamos encontrar todos os clientes que possuem conta e empréstimo em um banco.



```
SELECT DISTINCT nome_cliente
FROM devedor
WHERE nome_cliente IN (select nome_cliente
                        from depositante)
```

Observem que utilizamos acima o construtor **IN**, ele testa se um valor existe no outro subconjunto. Outros construtores importantes são o **UNIQUE** que testa se a subconsulta tem alguma tupla repetida no seu resultado; o **EXISTS** que retorna o valor TRUE se a subconsulta usada como argumento é “não-vazia”.

É possível ainda usar a palavra-chave **NOT** em conjunto com os operadores EXISTS e IN. Nestes casos o operador **NOT IN** verifica se o valor não existe em outro subconjunto. Semelhantemente podemos usar o NOT EXISTS que retorna um valor booleano para aceitação ou não da tupla em questão caso o argumento da subconsulta seja vazio.

Precisamos ainda entender as palavras chave **ALL**, **ANY** e **SOME**. A norma padrão não trata da palavra SOME, mas ela funciona da mesma forma que a ANY. A utilização dessas palavras serve para comparar o valor da

subconsulta externa com todos os valores da subconsulta interna. É possível, por exemplo, saber se um valor é maior que todos os valores da subconsulta. Vejam abaixo.

```
SELECT Name
FROM Product
WHERE ListPrice     ALL
      (SELECT ListPrice
       FROM Product
       GROUP BY ProductSubcategoryID);
```

Uma classificação para as subconsultas é se eles retornam uma ou múltiplas linhas. Algumas dicas devem ser levadas em consideração quando estamos elaborando subconsultas: (1) coloque subconsultas entre parênteses; (2) subconsultas devem ocupar o lado direito das comparações condicionais para facilitar a legibilidade; e (3) use operadores de única linha com subconsultas de uma **única linha** (`=, >, >=, <, <=, <>`) e operadores de **múltiplas linhas** (ALL e ANY) com subconsultas de várias linhas.

### 1.3.4. Funções agregadas (GROUP BY e HAVING)

Vamos conhecer agora as funções agregadas que operam sobre conjuntos de valores de uma coluna de uma relação e retornam um valor para cada conjunto, são elas: COUNT (), SUM (), MAX (), MIN (), AVG (). Quando utilizamos funções agregadas devemos utilizar o GROUP BY para os atributos na cláusula SELECT que não aparecem como parâmetros nas funções agregadas e a opção HAVING para predicados que são aplicados após a formação dos grupos.

Outra opção de subconsultas que retornam uma **única linha** é quando usamos funções de agregação como MIN e MAX. Vejam o exemplo abaixo para entendermos melhor a ideia:

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary = 2500
      (SELECT MIN(salary)
       FROM employees);
```

Vejamos uma questão sobre o assunto:



#### 3. Ano: 2017 Banca: FCC Órgão: TRT-11 Cargo: Técnico Judiciário de TI – Q. 44

Considere que no TRT exista, em um banco de dados, a tabela TRAB que possui como campos: nome, sexo, salario de vários trabalhadores. Um Técnico foi

solicitado a escrever um comando SQL para obter a média salarial dos trabalhadores do sexo FEMININO. O comando correto é:

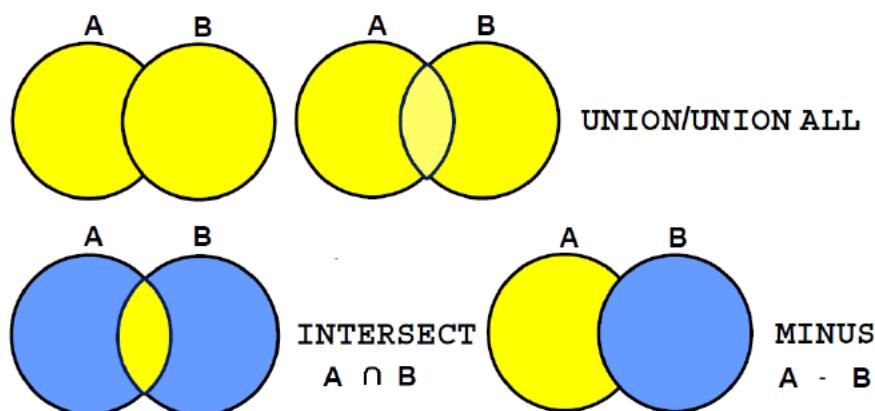
- (A) SELECT sexo="FEMININO" FROM TRAB WHERE AVG(salario);
- (B) SELECT sexo, AVG(salario) as MediaSalarial FROM TRAB GROUP BY sexo;
- (C) SELECT AVG(salario) FROM TRAB WHERE sexo='FEMININO';
- (D) SELECT sexo, AVG(salario) FROM TRAB GROUP BY sexo="FEMININO";
- (E) SELECT \* FROM TRAB WHERE sexo='FEMININO' as AVG(salario);

**Comentário:** Vejam que a questão pede apenas uma informação: a média salarial dos trabalhadores do sexo feminino. Desta forma, basta usarmos a função agregado que retorna a média dos valores não nulos de uma coluna (AVG) e restringirmos nossa consulta na cláusula WHERE pelos empregados do sexo feminino (sexo='FEMININO'). Sendo assim, podemos encontrar nossa resposta na alternativa C.

**Gabarito:** C.

### 1.3.5. Operações de conjuntos

As operações de conjuntos **UNION**, **INTERSECT** e **EXCEPT** operam em relações e correspondem às operações  $\cup$ ,  $\cap$  e  $-$  (diferença) da álgebra relacional. Estas operações eliminam as duplicatas. Se desejarmos obter as repetições, devemos explicitar através da forma **UNION ALL**, **INTERSECT ALL** e **EXCEPT ALL**.



É interessante perceber que a união é feita sobre o resultado de duas relações, ou seja, é possível duas consultas sobre tabelas distintas passarem por uma operação de união. É importante entender, porém, que os atributos das duas relações que participam da operação devem operar sobre os mesmos domínios. Vejamos um exemplo. Suponha que você queira encontrar todos os clientes que possuam um empréstimo, uma conta ou ambos.

```
(SELECT nome_cliente FROM depositante ) UNION
(select nome_cliente FROM devedor) SELECT Name
```

### 1.3.6. Junção

Por fim vamos falar da composição de relações utilizando junção ou JOIN. As operações de junção tomam duas relações e retornam como resultado outra relação. São normalmente usadas na cláusula from e devem ser feitas seguindo uma condição e um tipo de junção. A **condição de junção** define quais tuplas das duas relações apresentam correspondência, e quais atributos serão apresentados no resultado de uma junção. O **tipo de junção** define como as tuplas em cada relação que não possuem nenhuma correspondência (baseado na condição de junção) com as tuplas da outra relação devem ser tratadas.

Observem a figura com as condições e os tipos de junção abaixo:



Tipos de junção
<b>inner join</b>
<b>left outer join</b>
<b>rigth outer join</b>
<b>full outer join</b>

Condições de junção
<b>natural</b>
<b>on &lt;predicate&gt;</b>
<b>using (A<sub>1</sub>, A<sub>2</sub>,..., A<sub>n</sub>)</b>

Quando utilizamos a condição de junção **natural** o SQL basicamente vai utilizar os atributos das duas relações que possuem os mesmos nomes. Caso não deseje utilizar todos os atributos com nomes iguais você pode restringi-los utilizando a condição **using** passando como argumentos apenas os atributos que você deseja.

Vejam um exemplo abaixo do uso do NATURAL JOIN E do USING:

```
SELECT department_id, department_name,
       location_id, city
  FROM departments
NATURAL JOIN locations ;
```

```
SELECT employee_id, last_name,
       location_id, department_id
  FROM employees JOIN departments
USING (department_id) ;
```

Apenas uma consideração importante você não deve usar qualidficadores para as colunas que serão utilizadas na cláusula USING. Isso pode ocasionar um erro na execução da requisição (ORA-25154).

Após esse breve resumo da operação de SELECT, passaremos agora para analisar os demais comandos DML.

### 1.3.7. INSERT, UPDATE e DELETE

O comando INSERT INTO é utilizado para inserir novos registros em uma tabela. Sua sintaxe permite que você defina os valores para colunas de duas formas. Na primeira você não especifica os nomes das colunas nas quais os valores serão inseridos. Neste caso o SGBD vai relacionar os valores na mesma ordem em que foram definidos no comando CREATE. A segunda forma seria definir explicitamente os nomes das colunas e os valores. Vejam as duas opções a seguir:



**TOME NOTA!**

```
INSERT INTO table_name
VALUES (value1,value2,value3,...);

INSERT INTO table_name (column1,column2,column3,...)
VALUES (value1,value2,value3,...);
```

No ORACLE, se você quiser passar como parâmetro o valor corrente da data e hora é possível fazer uso da função SYSDATE. Outra possibilidade é usar como valores do comando INSERT uma subconsulta. Neste caso você não precisa usar a cláusula VALUES e o número de coluna da subconsulta deve ser igual ao número de colunas do INSERT. Vejamos um exemplo para ajudar na fixação do assunto:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows inserted

Outro comando que compõe a linguagem é o UPDATE. Utilizado para atualizar registros em uma tabela. Ele basicamente define na sua cláusula WHERE quais linhas ou registros devem ser atualizados, caso nenhuma verificação seja feita todas as linhas serão atualizadas. As mudanças a serem executadas são definidas na cláusula SET. Vejam a sintaxe do comando abaixo:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE some_column=some_value;
```

Valores para um conjunto de linhas específicas são atualizados ou modificados se você definir os critérios na cláusula WHERE. Se você quiser atribuir à coluna o valor NULL basta usar SET nome\_da\_coluna = NULL. Lembre-se que caso nenhum critério for definido na cláusula WHERE, todas as linhas são atualizadas. Veja alguns exemplos do comando UPDATE:

```
UPDATE employees
SET department_id = 50
WHERE employee_id = 113;
1 rows updated
```

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated
```

Vejamos agora algumas informações sobre o DELETE. A instrução DELETE é usada para deletar linhas de uma tabela. Veja que se nenhuma condição for definida na cláusula WHERE do comando todas as linhas serão removidas da tabela. A sintaxe do comando é apresentada a seguir:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

Outro comando definido como **DDL** mas que tem como funcionalidade remover de forma rápida e fácil todas as linhas de uma tabela é o comando TRUNCATE. Ele remove todo conteúdo da tabela, zera os índices e as sequências associadas. Para tal basta utilizar o comando DDL: TRUNCATE TABLE nome\_tab;

```
TRUNCATE [ TABLE ] [ ONLY ] name [ * ] [, ... ]
[ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE | RESTRICT ]
```

Podemos observar acima o comando TRUNCATE com a sintaxe específica do POSTGRES. Nele é possível remover um conjunto de tabelas ao mesmo tempo. O primeiro parâmetro permite reiniciar ou não as sequências associadas as colunas das tabelas cujos dados serão removidos, usando respectivamente o **restart identity** e **continue identity**. O outro parâmetro, **cascade** ou **restrict**, vai aplicar o comando truncate as tabelas que tem relacionamento ou chaves estrangeiras referenciando uma das tabelas removidas.

## 1.4. DDL Complementos: VIEW

Outro objeto SQL que podemos criar dentro dos nossos bancos de dados são as VIEWS. A view é um comando SQL que é armazenado no banco de dados e possui um nome associada a ela. Eles têm algumas funções básicas. A primeira é facilitar a visualização dos dados dispersos em diversas tabelas tornando-os mais natural ou intuitivo ao entendimento humano.

Outra função importante para a view está relacionada a segurança dos dados. É possível restringir o acesso aos campos e às colunas de uma tabela por meio de uma view. Desta forma o usuário teria visão apenas a parte dos dados ou informações. Esse grupo de informações deve ser compatível com as funções e necessidades de acesso do usuário.

Outra opção para o uso de view é sumarizar dados de diferentes tabelas gerando relatórios. Se pensarmos no INFORMATION\_SCHEMA que tratamos no início da aula, ele geralmente é composto por um conjunto de visões que trazem os dados referentes as tabelas dos esquemas do seu banco de dados.

Vejamos abaixo dois exemplos do uso de **VIEWS**. Lembrando que ela pode ser criada sobre uma ou múltiplas tabelas. Observe que o comando basicamente inclui a sintaxe CREATE VIEW nome AS antes de uma consulta ao banco de dados.

```
CREATE VIEW web_designers AS
SELECT mc.first_name, mc.last_name, mc.phone, mc.email
FROM my_contacts mc
NATURAL JOIN job_desired jd ← This could also have been an INNER
                                JOIN using:
                                ON mc.contact_id = jd.contact_id.
WHERE jd.title = 'Web Designer';
```

```
CREATE VIEW tech_writer_jobs AS
SELECT title salary, description, zip
FROM job_listings
WHERE title = 'Technical Writer';
```

Para visualizarmos os dados de uma visão, basta escrevermos um comando SELECT sobre ela, vejam o exemplo sobre a view **web\_designers** criada acima.

```
SELECT * FROM web_designers;
```

Here's the name  
of our view.

A view é considerada uma tabela virtual porque ela só existe durante o período que você está utilizando a mesma. Todas as operações que são feitas sobre a tabela podem ser feitas em uma VIEW, mas a tabela é virtual, e na teoria não deve ser armazenada no banco de dados.

Como falamos é possível usar os comandos DML (Select, Insert, Update e Delete) sobre uma visão. Essas modificações podem ou não ter seus efeitos sobre os dados armazenados nas tabelas subjacentes, ou seja, associadas a visão. Uma visão que permite a atualização da tabela subjacente tem uma associação direta entre as linhas da visão e as linhas da tabela.

O padrão SQL/ANSI define algumas regras que indicam se a visão pode ser atualizada. Primeiramente, a visão deve ser construída com base em apenas uma tabela. As cláusulas GROUP BY, HAVING e SELECT DISTINCT não podem estar presentes. Não podem existir nenhuma função de agregação ou colunas calculadas (atributos derivados).



Se você está usando valores agregados na visão (por exemplo, SUM, COUNT e AVG), você não tem a permissão de alterar os dados. Da mesma forma, se na sua view tiver as palavras chave GROUP BY, DISTINCT, ou HAVING, também não é permitido fazer alterações nos dados.

Por implicação das regras acima, uma coluna da view deve ser a chave da tabela subjacente. Enfim, nós precisamos ter certeza de que a partir de uma linha da visão conseguimos referenciar uma linha da tabela.

É possível ainda usar a cláusula WITH CHECK OPTION ao final do comando de definição da VIEW. Nestes casos qualquer INSERT ou UPDATE que seja executado sobre a visão irá verificar se os dados inseridos ou modificados estão de acordo com as restrições descritas na cláusula WHERE.

Por fim é possível apagar a visão quando você não precisar mais dela por meio do comando DROP VIEW nome\_da\_visao.

## 1.5. SQL Embutido

O padrão SQL define que a SQL será embutida em uma variedade de linguagens de programação, como Pascal, PL/I, Fortran, C e Cobol. A linguagem na qual são embutidas consultas SQL é chamada linguagem hospedeira, e as estruturas SQL permitidas na linguagem hospedeira são denominadas SQL embutida.

EXEC SQL é usado para identificar os pedidos em SQL embutida para o pré-processador EXEC SQL <comando SQL embutido> END EXEC. Suponha que temos na linguagem hospedeira uma variável chamada total e que desejamos encontrar os nomes e cidades dos clientes dos clientes que tenham mais de um total em dólares em qualquer conta. Podemos escrever essa consulta como segue:



```
EXEC SQL
declare c cursor for
select nome_cliente,cidade_cliente
from deposito,cliente
where deposito.nome_cliente = cliente.nome_cliente
and depósito.saldo > :total
END-EXEC
```

O SQL embutido permite ainda o uso de alguns comandos. O comando **open** faz com que a consulta seja avaliada:

```
EXEC SQL open c END-EXEC
```

O comando **fetch** determina os valores de uma tupla que serão colocados em variáveis da linguagem de host.

```
EXEC SQL fetch c into :cn :an END-EXEC
```

Pode-se utilizar um laço para processar cada tupla do resultado. Uma variável na área de comunicação do SQL indica que não há mais tupla a ser processada.

O comando **close** faz com que o sistema de banco de dados remova a relação temporária mantida para o resultado da consulta.

```
EXEC SQL close c END-EXEC
```

## 1.6. Transações em SQL

Uma transação é um conjunto de instruções SQL que realizam um único trabalho. O exemplo clássico de transação é a transferência de dinheiro de uma conta para outras. Primeiro é necessário saber o saldo da conta X, depois retirar o dinheiro desta conta e creditar em outra conta Y. Essas três ações se caracterizam um trabalho que deve ser executado em um único bloco.

Durante a transação todos os passos devem ser executados. Isso nos leva ao conceito de **atomicidade** da transação. Caso um passo não possa ser executado o banco de dados deve ser posto no seu estado inicial válido. Isso garante a **consistência** da base após a execução de uma transação.

Além da atomicidade e consistências, são consideradas propriedades de uma transação o **isolamento** e a **durabilidade**. O primeiro se refere ao fato que uma transação no deve enxergar os dados e a execução de outras transações. O segundo afirma que caso uma transação termine com sucesso seus dados ou suas modificações feitas sobre a base de dados tornam-se persistentes.

Uma *transaction* é um mecanismo que podemos utilizar para manter a integridade e a consistência dos dados. SQL nos ajuda a gerenciar transações. SQL padrão definiu transações logo no início da criação do modelo e vem

aprimorando o conceito durante iterações subsequentes. De acordo com a norma, uma transação é iniciada pelo SGBDR, e continua até que uma instrução de COMMIT ou ROLLBACK é emitida.

Os detalhes foram deixados para o SGBD implementar. Os comandos de gerenciamento de transações do SQL padrão estão listados na tabela abaixo:



Comando	Descrição
START (BEGIN) TRANSACTION	Inicializa uma transação SQL e seta as suas características.
SET TRANSACTION	Determina as propriedades da próxima transação SQL para o SQL Agent
SET CONSTRAINTS	Se a transação SQL estiver executando, estabelece o modo de restrições para a transação SQL na sessão corrente. Se não existe nenhuma transação em andamento na sessão, determina ao SQL Agent o modo de execução para a próxima transação.
SAVEPOINT	Estabelece um <i>savepoint</i> , ponto intermediário da transação para o qual o <i>rollback</i> deve retornar em caso de falha.
RELEASE SAVEPOINT	Destroi um <i>savepoint</i>
COMMIT	Termina a transação SQL corrente com um <i>commit</i>
ROLLBACK	Termina a transação corrente com um <i>rollback</i> , ou desfaz todas as modificações até o último <i>savepoint</i> .

O padrão SQL utiliza de forma implícita o START TRANSACTION, com uma instrução de COMMIT explícita, nos casos em que todas as unidades lógicas de trabalho de uma transação são concluídas com sucesso. Podemos ainda ter a instrução de ROLLBACK quando mudanças ainda não efetivadas precisam ser desfeitas devido a alguma exceção ou erro.

As implementações dos SGBDs para controle de transação são um pouco diferentes para cada um deles, contudo a análise detalhada destes comandos

foge do escopo deste curso. Vamos passar agora para uma rápida explicação dos níveis de isolamento de transações. Mais detalhes sobre esse assunto serão vistos em aulas específicas sobre transações e controle de concorrência.

Quando executamos várias transações de forma concorrente sobre o mesmo banco de dados podemos limitar o acesso as informações para impedir que elas sejam vistas por uma transação no momento que estão sendo usadas por outra transação. São quatro os níveis de isolamento definidos pelo SQL ANSI: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ e SERIALIZABLE.

No primeiro caso READ UNCOMMITTED é permito ler dados que ainda não sofreram o COMMIT de outra transação X que ainda está em execução. Isso acaba possibilitando a existência do problema de leitura suja ou dirty read. Esse problema ocorre se a transação X sofrer ROLLBACK. Neste caso, qualquer outra transação que tiver feito uma consulta sobre o dado modificado por X terá feito uma leitura suja.

Para resolver esse problema surge o próximo nível de isolamento conhecido como READ COMMITTED. Neste caso uma transação Y só pode ler os dados modificados pela transação X após o COMMIT. Agora ainda permanecemos com outro problema denominado leitura não-repetível. Imagine que Y precisa ler os dados modificados por X duas vezes durante a transação. Na primeira leitura Y ler os dados modificados por X, contudo, antes da segunda leitura, outra transação Z faz uma nova modificação sobre os dados. Veja que Y faz duas leituras sobre o mesmo dado durante a transação e eles possuem valores distintos.

A solução para os problemas da leitura não-repetível está no REPEATABLE READ. Neste caso a transação Y vai ler os mesmos dados, pois não será permito a transação Z fazer modificações sobre eles durante a execução de Y. Agora contamos com um último problema, qual seja, a possibilidade da existência de registros fantasmas. Neste caso Y faz duas leituras sobre um conjunto de dados ou tuplas de uma relação que satisfação a uma condição descrita na cláusula WHERE. Suponha que entre a primeira e a segunda leitura seja inserido um registro que também satisfaz a essa condição de busca. Esse registro é considerado um fantasma!

Para resolver todos esses problemas, temos o último nível de isolamento conhecido como SERIALIZABLE. A ideia é executar concorrentemente apenas as transações que produzam o mesmo resultado caso fossem executadas em série.

Além do nível de isolamento é possível determinarmos o modo de acesso de uma transação que pode ser READ ONLY ou READ WRITE e o amanho da área de diagnóstico que especifica um valor inteiro  $n$ , indicando o número de posições que podem ser manipuladas simultaneamente na área de diagnóstico. Essas condições fornecem informações sobre as condições de execução (erros e

exceções), ao usuário ou ao programa, para a maior parte das declarações SQL executadas mais recentemente.

Para executar uma transação usando SQL podemos utilizar a seguinte sintaxe:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE
EXEC SQL INSERT INTO EMPREGADO (PNOME, UNOME, SSN, DNO, SALARIO)
VALUES ('Thiago', 'Cavalcanti', 000457878, 2, 12000);
EXEC SQL UPDATE EMPREGADO
    SET SALARIO = SALARIO *1,1 WHERE DNO =2;
EXEC COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ...;
```

## 1.7. Index (DDL)

Índices são estruturas opcionais associadas às tabelas. Logo, não existe índice se não estiver associado a uma tabela. É possível que a criação dos índices seja feita de forma explícita para melhorar o desempenho na execução de comandos sobre uma tabela. Semelhante a um índice de um livro, os índices em bancos de dados podem prover um caminho de acesso mais rápido para os dados em uma tabela.

Os índices, quando usado apropriadamente, vão permitir uma redução das operações de I/O sobre o disco. A criação de índices pode ser feita sobre um ou mais campos da sua tabela. Essa possibilidade nos leva a primeira taxonomia possível para classificação dos índices, que pode ser simples ou composto.

A criação do índice pode ser feita sobre uma tabela em branco ou uma tabela que já contenha tuplas armazenadas.

Outra classificação para índice está relacionada com uma associação entre os valores existentes no arquivo de índice e os valores armazenados em uma tabela sobre os quais os índices são estruturados. Quando essa relação é de um para um dizemos que o índice é denso. Quando essa relação não é direta entre a chave de pesquisa e a quantidade de valores possíveis então dizemos que o índice é esparsa.

Quando tratamos de SQL precisamos pensar que a criação do índice é um comando DDL. Ele vai melhorar o desempenho das consultas que recuperam

registros por meio do comando SELECT, desde que a (s) coluna (s) que compõe o índice estejam presentes na condição de busca. Por outro lado, a utilização de índices vai degradar o desempenho dos comandos UPDATE, INSERT e DELETE. A utilização destes comandos implica na atualização, mesmo que transparente, do arquivo de índices.

A sintaxe básica do comando descrita na norma SQL/ANSI é seguinte:

```
CREATE INDEX ix_name ON table_name (col_name);
```

Para destruir ou apagar um índice basta usar o comando:

```
DROP INDEX ix_name;
```

Apenas para terminar o assunto, gostaria de fazer uma consideração histórica sobre a possibilidade do uso da cláusula UNIQUE no comando de criação dos índices. Essa opção usada para garantir a unicidade do valor da coluna, implicitamente criada para a (s) coluna (s) da chave primária. Porém foi descontinuada na maioria dos SGBDs, agora usamos a restrição de UNIQUE para coluna.

## 1.8. Extensões procedurais

Quando falamos em procedimentos armazenados do ponto de vista de SQL podemos invocar três diferentes estruturas: as stored procedures, os triggers e as user defined functions (UDFs). Esses padrões estão descritos no SQL/PSM (Persistent Stored Module) uma parte da documentação oficial que tenta incorporar esses conceitos, falamos anteriormente dela no início da aula.

Desde a sua formulação original no padrão SQL-86, um dos principais inconvenientes que impediram as pessoas de usar o SQL foi a sua falta de fluxo de instruções de controle. Foi então que o SQL/PSM foi incluído no padrão SQL, você não poderia ramificar a ordem sequencial de execução sem utilizar outra linguagem como C ou BASIC. SQL/PSM introduz o fluxo tradicional de estruturas de controle que existem em outras linguagens, permitindo, assim, que programas SQL possam executar funções necessárias sem a utilização de outras linguagens.

Esses recursos processuais definem, entre outros componentes, a criação de rotinas SQL que podem ser invocada explicitamente como procedimentos e funções.

Os procedimentos armazenados residem no servidor de banco de dados, em vez de executar no cliente - onde todos os procedimentos eram localizados antes do SQL/PSM. Depois de definir um procedimento armazenado, você pode invocá-lo com uma instrução CALL. Mantendo o procedimento armazenado no servidor é possível reduzir o tráfego da rede, acelerando assim o desempenho. O

único tráfego que precisa ser feito a partir do cliente ao servidor é a instrução CALL. Você pode criar este procedimento da seguinte maneira:

```
EXEC SQL
  CREATE PROCEDURE MatchScore
    ( IN result CHAR (3),
      OUT winner CHAR (5) )
  BEGIN ATOMIC
    CASE result
      WHEN '1-0' THEN
        SET winner = 'white' ;
      WHEN '0-1' THEN
        SET winner = 'black' ;
      ELSE
        SET winner = 'draw' ;
    END CASE
  END ;
```

Depois de ter criado um procedimento armazenado como o descrito neste exemplo, você pode invocá-lo com uma instrução CALL semelhante à seguinte declaração:

```
CALL MatchScore ('Kasparov', 'Karpov', '1-0', winner);
```

Os três primeiros argumentos são parâmetros de entrada que são alimentam o procedimento MatchScore. O quarto argumento é o parâmetro de saída que o procedimento usa para retornar o resultado da execução da rotina. Neste caso, ele retorna 'white'.

A função armazenada ou UDF é semelhante em muitos aspectos a um procedimento armazenado. Coletivamente, os dois são referidos como rotinas armazenadas. Eles são diferentes em vários aspectos, incluindo a forma como são chamados ou executados. Um procedimento armazenado é chamado por meio da instrução CALL, e uma função armazenada é chamado como uma chamada de função, que pode substituir um argumento de uma instrução SQL. A seguir temos um exemplo de uma definição de função, seguido por um exemplo de uma chamada para essa função:

```
CREATE FUNCTION PurchaseHistory (CustID)
RETURNS CHAR VARYING (200)
BEGIN
    DECLARE purch CHAR VARYING (200)
        DEFAULT '' ;
    FOR x AS SELECT *
        FROM transactions t
        WHERE t.customerID = CustID
    DO
        IF a <> ''
            THEN SET purch = purch || ', ' ;
        END IF ;
        SET purch = purch || t.description ;
    END FOR
    RETURN purch ;
END ;
```

Esta definição de função cria uma lista de compras feitas por um cliente, baseada em um número de cliente especificado, extraída da tabela TRANSACTIONS e separada por vírgulas. A instrução UPDATE a seguir contém uma chamada de função de PurchaseHistory que insere o histórico mais recente de compras do cliente de número 314259 em seu registro na tabela CLIENTE:

```
SET customerID = 314259 ;
UPDATE customer
    SET history = PurchaseHistory (customerID)
    WHERE customerID = 314259 ;
```

Até este ponto da aula, você aprendeu a criar uma série de objetos de esquema que você pode acessar ou invocar usando instruções SQL. Por exemplo, você aprendeu como criar tabelas, visões e rotinas. Em cada caso, uma vez que você crie esses objetos, você precisa tomar algum tipo de ação para interagir diretamente com eles, como executar uma instrução SELECT para recuperar dados de uma tabela ou usando uma instrução CALL para invocar um procedimento.

No entanto, SQL oferece suporte a objetos que executam ações automaticamente. Esses objetos de esquema, que são conhecidos como gatilhos, eles respondem às modificações feitas nos dados em uma tabela. Se uma modificação especificada é feita, o gatilho é invocado automaticamente, ou disparado, causando uma ação adicional. Como resultado, você nunca vai invocar diretamente uma ação definida no gatilho. O trigger vai implicitamente fazer a chamada e execução dos seus comandos. Abaixo, vamos explorar gatilhos e como eles são usados quando os dados de uma tabela são modificados. A sintaxe básica para a criação de um gatilho é definida abaixo.

```
CREATE TRIGGER <trigger name>
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT | DELETE | UPDATE [ OF <column list> ] }
ON <table or view name> [ REFERENCING <alias options> ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( <search condition> ) ]
<triggered SQL statements>
```

Vamos dar uma olhada em cada linha. A primeira linha é bastante simples. Você simplesmente fornecer um nome para o gatilho após as palavras-chave CREATE TRIGGER. Na segunda linha, você deve designar se o gatilho é chamado antes, depois ou em vez da modificação de dados especificada na instrução SQL que fez o gatilho para disparar. Por exemplo, se você definiu um gatilho de inserção, você pode especificar se as instruções SQL açãoadas sejam executadas antes (BEFORE) dos dados serem inseridos na tabela. A opção BEFORE é particularmente útil quando uma das tabelas é configurada com uma restrição de integridade referencial.

Na terceira linha da sintaxe, você especifica se o gatilho é de inserção, exclusão ou atualização. Se você definir um TRIGGER de atualização, você tem a opção de aplicar o gatilho para uma ou mais colunas específicas. Se mais de uma coluna for especificado, você deve separar os nomes das colunas com vírgulas. Na próxima linha (4) da sintaxe, você deve especificar uma cláusula ON, que inclui o nome da tabela ou visão sobre a qual o gatilho é aplicado. O gatilho pode ser aplicado a apenas uma tabela ou vista. Tenha em mente que, gatilhos de BEFORE e AFTER podem ser aplicados somente a tabelas, enquanto que o gatilho de INSTEAD OF pode ser aplicado apenas a visões.

Até este ponto, toda a sintaxe que aprendemos é obrigatória, exceto o nome das colunas em definições de gatilhos de atualização, que é opcional. No entanto, as próximas cláusulas não são obrigatórias, mas elas adicionam recursos importantes para o seu gatilho.

A primeira destas cláusulas é a cláusula REFERENCING. Esta cláusula permite que você especifique como os dados, que utilizam no contexto de execução do gatilho, são referenciados dentro da cláusula WHEN ou nas instruções SQL. Você pode usar uma referência para os valores novos ou antigos das colunas que estão sendo modificadas pelo comando que dispara o gatilho. Veja um exemplo abaixo:



```
CREATE TRIGGER UPDATE_TITLE_COSTS
  AFTER UPDATE ON TITLES_IN_STOCK
  REFERENCING NEW ROW AS New
  FOR EACH ROW
  WHEN ( New.INVENTORY > 20 )
  BEGIN ATOMIC
    UPDATE TITLE_COSTS c
      SET RETAIL = ROUND(RETAIL * 0.9, 2)
    WHERE c.CD_TITLE = New.CD_TITLE;
  END;
```

A próxima linha de sintaxe contém a cláusula FOR EACH, que inclui duas opções: ROW ou STATEMENT. Se você especificar ROW, o gatilho é invocado sempre que uma linha for inserida, atualizada ou excluída. Se você especificar STATEMENT, o gatilho é chamado somente uma vez para cada instrução de modificação de dados aplicável que é executada, não importa quantas linhas foram afetadas. Se você não incluir esta cláusula em sua definição de gatilho, a opção STATEMENT é assumida.

Ao avançar na sintaxe temos a cláusula WHEN que é opcional e não é suportado para gatilhos de INSTEAD OF. A cláusula WHEN permite definir um critério de pesquisa que limita o escopo do gatilho criando uma condição para o mesmo ser invocado. A cláusula WHEN é semelhante à cláusula WHERE de uma instrução SELECT. Você especificar um ou mais predicados que definem uma condição de pesquisa. Se a cláusula WHEN for avaliada como verdadeira, o gatilho é acionado; caso contrário, nenhuma ação será tomada pelo gatilho. No entanto, isso não afeta a instrução de modificação de dados inicial que foi executada contra a tabela em questão; somente as instruções SQL acionadas na definição do gatilho são afetadas.

Finalmente, o último componente que seu CREATE TRIGGER deve incluir é um ou mais instruções SQL que são executados quando o gatilho é chamado. Veja no exemplo mostrado anteriormente uma instrução de UPDATE que é executada pelo gatilho quando a condição definida pela cláusula WHEN é avaliada como verdadeira.

## 1.9. Segurança – Utilizando a DCL

As instruções SQL que você usa para criar bancos de dados formam um grupo conhecido como a linguagem de definição de dados (DDL). Depois de criar um banco de dados, você pode usar outro conjunto de instruções SQL - conhecidos coletivamente como Data Manipulation Language (DML) - para adicionar, alterar e remover os dados do banco de dados. SQL inclui declarações adicionais que não se enquadram em nenhuma dessas categorias.

Os programadores, por vezes, referem-se a estas declarações coletivamente como a Linguagem de Controle de Dados (DCL). Declarações ou instruções DCL protegem o banco de dados contra acessos não autorizados, a

partir da interação prejudicial entre vários usuários de banco de dados. Neste bloco, tratamos da proteção contra acesso não autorizado.

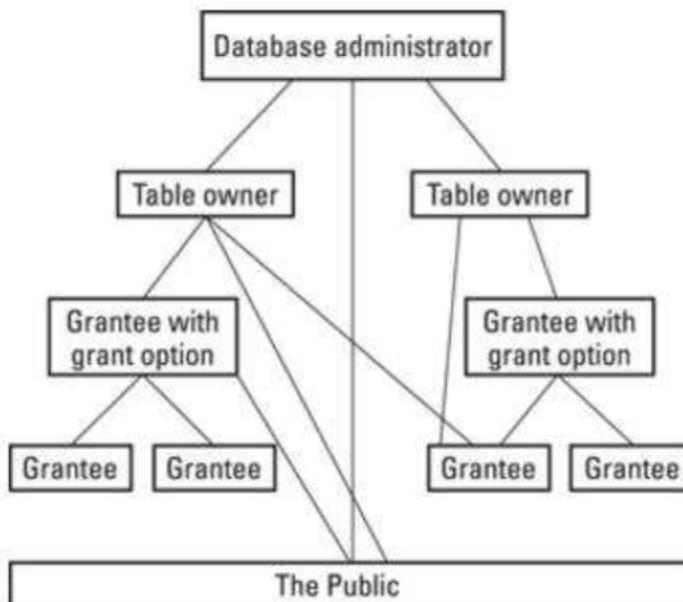
O SQL fornece acesso controlado a nove funções de gerenciamento de banco de dados: INSERT, SELECT, UPDATE, DELETE, REFERENCES, USAGE, UNDER, TRIGGER e EXECUTE.

Os quatro primeiros estão relacionados a instruções de manipulação de dados. O REFERENCES envolve a autorização do uso da integridade referencial em uma tabela que depende de outra tabela. A palavra-chave USAGE diz respeito a domínios, conjuntos de caracteres, *collations* e *translations*. Usamos o UNDER quando estamos tratando de tipos definidos pelo usuário. O TRIGGER garante a autorização para executar o comando quando um evento predeterminado acontece. O EXECUTE roda uma determinada rotina.

Quando possuímos permissões destes tipos sobre os objetos podemos executar as funções autorizadas sobre eles.

### 1.9.1. A hierarquia do privilégio de acesso

Para entendermos o que essa hierarquia significa vamos observar a figura abaixo. Na maioria das instalações, existe uma hierarquia de privilégios de usuário, em que o DBA está no mais alto nível e o PUBLIC no menor.



Em termos de rede, "PUBLIC" consiste de todos os usuários que não são usuários privilegiados (isto é, nem DBAs ou proprietários de objetos) e para quem um usuário privilegiado não concedeu direitos de acesso especificamente. Se um usuário concede determinados direitos de acesso privilegiados ao PUBLIC, então todos que podem acessar o sistema ganham esses direitos.

### 1.9.2. Garantindo privilégios aos usuários

O DBA, em virtude de sua posição, tem todos os privilégios em todos os objetos do banco de dados. O proprietário de um objeto também tem todos os privilégios em relação a esse objeto – o próprio banco de dados é um objeto. Ninguém mais tem qualquer privilégio em relação a qualquer objeto, a menos que alguém que já tem esses privilégios (e autoridade para passá-los) concede especificamente os privilégios. Você concede privilégios para alguém usando a instrução GRANT, que tem a seguinte sintaxe:

```

GRANT privilege-list
ON object
TO user-list
[WITH HIERARCHY OPTION]
[WITH GRANT OPTION]
[GRANTED BY grantor];
    
```

Neste comando a lista de privilégios pode ser uma lista separada por vírgulas ou ALL PRIVILEGES. Este último vai garantir o privilégio em todo o conjunto dos nove privilégios que tratamos anteriormente, são eles:

```

SELECT
| DELETE
| INSERT [(column-name [, column-name]...)]
| UPDATE [(column-name [, column-name]...)]
| REFERENCES [(column-name [, column-name]...)]
| USAGE
| UNDER
| TRIGGER
| EXECUTE
    
```

Os objetos definidos no comando podem ser os seguintes:

```

[ TABLE ] <table name>
| DOMAIN <domain name>
| COLLATION <collation name>
| CHARACTER SET <character set name>
| TRANSLATION <transliteration name>
| TYPE <schema-resolved user-defined type name>
| SEQUENCE <sequence generator name>
| <specific routine designator>
    
```

Para finalizar o comando a lista de usuários pode ser um conjunto separado por vírgulas ou PUBLIC. E o garantidor (grantor) é o usuário corrente (CURRENT\_USER) ou o papel (CURRENT\_ROLE). Vamos aproveitar que tratamos de ROLES para descrever um pouco mais sobre suas características.

### 1.9.3. Roles

Um nome de usuário é um tipo de identificador de autorização, mas não é o único. Ele identifica uma pessoa (ou um programa) autorizada a executar uma ou mais funções em uma base de dados. Em uma grande organização com muitos usuários, a concessão de privilégios para cada funcionário individualmente pode ser tediosa e demorada. SQL resolve este problema introduzindo a noção de papéis.

Um papel, identificado por um nome, é um conjunto de zero ou mais privilégios que podem ser concedidos a várias pessoas, pois todas elas exigem o mesmo nível de acesso ao banco de dados. Por exemplo, todos os que desempenham o papel contador tem os mesmos privilégios. Esses privilégios são diferentes dos concedidos às pessoas que têm a função de balonista.

Essa não é uma característica mencionada na última versão da especificação SQL, mas está disponível em todas as implementações de SGBDs. Verifique a documentação antes de tentar usar papéis.

Você pode criar um papel usando uma sintaxe semelhante à seguinte:

```
CREATE ROLE balonista;
```

Depois de criar um papel, você pode atribuir pessoas ao papel com a instrução GRANT, da seguinte forma:

```
GRANT balonista TO Jose;
```

Você pode conceder privilégios a um papel exatamente da mesma maneira que você conceder privilégios para usuários.

### 1.9.4. WITH GRANT OPTION

O DBA pode conceder qualquer privilégio. Um proprietário de um objeto pode conceder quaisquer privilégios sobre esse objeto a qualquer pessoa ou role. Mas, os usuários que recebem privilégios desta forma não podem, por sua vez, conceder esses privilégios para outra pessoa. Esta restrição ajuda o DBA ou o proprietário da tabela a manter o controle sobre o objeto. Apenas os usuários que o DBA ou o proprietário do objeto atribui competência para executar a operação em questão pode fazê-la.

Do ponto de vista de segurança, colocar limites na capacidade de delegar privilégios de acesso faz muito sentido. Muitas vezes surgem, no entanto, situações em que os usuários precisam do poder de delegar ou repassar seus privilégios. O trabalho não pode sofrer uma parada brusca cada vez que alguém está doente, em férias, ou saiu para almoçar.

Você pode confiar a alguns usuários o poder de delegar seus direitos de acesso. Para repassar esse direito de delegação para um usuário, o GRANT usa a cláusula **WITH GRANT OPTION**. A declaração a seguir mostra um exemplo de como você pode usar essa cláusula:

```
GRANT UPDATE (BonusPct)
  ON BONUSRATE
  TO SalesMgr
  WITH GRANT OPTION;
```

Agora, o gerente de vendas pode delegar o privilégio UPDATE emitindo a seguinte declaração:

```
GRANT UPDATE (BonusPct)
  ON BONUSRATE
  TO AsstSalesMgr;
```

Após a execução desta declaração, qualquer pessoa com o papel de gerente de vendas assistente pode fazer alterações para a coluna BonusPct da tabela de BONUSRATE. Você faz uma troca entre segurança e conveniência quando você delega direitos de acesso a um suplente. O proprietário da tabela de BONUSRATE abandona o controle considerável na concessão do privilégio UPDATE para o gerente de vendas usando o WITH GRANT OPTION.

### 1.9.5. Removendo privilégios

Se você tem uma maneira de dar privilégios de acesso para as pessoas, você também deve ter uma forma de tirar os privilégios. As funções de trabalho das pessoas mudam, e com essas mudanças as suas necessidades de acesso de dados mudam. Digamos que um funcionário deixe a organização para trabalhar em um concorrente. Você provavelmente deve revogar todos os privilégios de acesso dessa pessoa - imediatamente.

SQL permite remover privilégios de acesso usando a instrução REVOKE. Esta declaração age como a instrução GRANT faz, exceto que ele tem o efeito inverso. A sintaxe para essa instrução é a seguinte:

```
REVOKE [GRANT OPTION FOR] privilege-list
  ON object
  FROM user-list [RESTRICT/CASCADE];
```

Você pode usar essa estrutura para revogar os privilégios especificados. A principal diferença entre a instrução REVOKE e a instrução GRANT é a presença da palavra-chave opcional RESTRICT ou CASCADE na instrução REVOKE.

Por exemplo, suponha que você usou a cláusula WITH GRANT OPTION quando concedeu determinados privilégios a um usuário. Eventualmente,

quando você deseja revogar os privilégios, você pode usar CASCADE na instrução REVOKE. Quando você revogar os privilégios de um usuário, desta forma, você também retira os privilégios de qualquer pessoa a quem essa pessoa tenha concedido privilégios.

Por outro lado, a declaração REVOKE com a opção RESTRICT funciona apenas se o beneficiário não delegou os privilégios especificados. Nesse caso, a instrução REVOKE revoga privilégios do beneficiário apenas. Mas, se o usuário passou os privilégios especificados, a instrução REVOKE com a opção RESTRICT não revogar nada – e, em vez disso, retorna um código de erro. Este é um aviso claro para você que você precisa descobrir para quem foram concedidos os privilégios pela pessoa cujos privilégios você está tentando revogar. Você pode ou não querer revogar os privilégios dessa pessoa.

Você pode usar uma instrução REVOKE com a cláusula opcional GRANT OPTION FOR de revogar apenas a opção de concessão de privilégios especificados ao mesmo tempo permitindo que o beneficiário a manter os privilégios para si mesmo. Se a cláusula GRANT OPTION FOR e a palavra-chave CASCADE estiverem presentes, você vai revogar todos os privilégios que o beneficiado concedeu, juntamente com o direito do beneficiário de conceder tais privilégios – funciona como se você nunca tivesse concedido a opção de concessão. Se a cláusula GRANT OPTION FOR e a cláusula RESTRICT estiverem presentes, uma de duas coisas acontece:

1. Se o beneficiário não concedeu a qualquer pessoa um dos privilégios que você está revogando, a instrução REVOKE executa e remove a capacidade do beneficiário de conceder privilégios.
2. Se o beneficiário já concedeu pelo menos um dos privilégios que você está revogando, a instrução REVOKE não é executada e retorna um código de erro.

## 2. Interfaces de utilização

Para ter acesso aos dados presentes no banco de dados existem diferentes interfaces que podem ser utilizadas pelos grupos de usuários. No cenário mais comum, é necessária uma conexão ao banco de dados. Essa facilidade permite que o software cliente ou aplicação comunique-se com o sistema de gerenciamento de banco de dados.

Os programas desenvolvidos em **Java**, por exemplo, comunicam-se com um banco de dados e manipulam seus dados utilizando a **API JDBC**. Um driver JDBC permite aos aplicativos Java se conectarem a um banco de dados em particular e manipularem a base de dados. Os SGBDs, geralmente, fornecem seus drivers JDBC e também outros diferentes drivers independentes.

O **Open Database Connectivity** (ODBC) e **Java Database Connectivity** (JDBC) são padrões para drivers que permitem aos programadores escreverem aplicativos de software de banco de dados de forma agnóstica. O que seria isso? Eles não se preocupam com o SGBD específico que seja utilizado, apenas com as interfaces.

Em outras palavras, ODBC e JDBC são padrões, ou um conjunto de regras recomendado para uma comunicação eficiente com um banco de dados. O fornecedor de banco de dados é responsável pela implementação e fornece os drivers que estão habilitados a seguir estas regras definidas pela interface.

Dependendo do seu ambiente e das operações que você deseja realizar sobre o banco de dados, você deve decidir por usar um driver ODBC ou um driver JDBC que vai se adaptar melhor às suas necessidades. Use o seguinte raciocínio para ajudá-lo a decidir:

**Use ODBC nativo** para ter um desempenho mais rápido nas importações e exportações de dados, ou ainda, nas importações e exportações de dados que exigem muita memória.

**Use JDBC** quando for necessário tratar a independência de plataforma, o que lhe permite trabalhar com qualquer sistema operacional (incluindo Mac e Linux) ou versão de driver. Use também quando as funções de banco de dados não suportarem a interface ODBC nativa (como *run stored procedure*). Por fim, use JDBC quando estiver trabalhando com tipos de dados complexos ou muito longos (por exemplo, LONG, BLOB, TEXT, etc.).

Falaremos a seguir um pouco sobre ODBC.

## 2.1. ODBC.

O ODBC (*Open Database Connectivity*) é uma API (*Application Programming Interface*) com ampla aceitação para acesso a bases de dados. É baseada nas especificações **Call-Level Interface (CLI)** do X/Open e ISO/IEC para APIs de acesso a bases de dados e usa SQL como linguagem de acesso.

A API permite aos programas, a partir do lado cliente, possam chamar o SGBD, desde que as máquinas clientes e o servidor tenham um software instalado. Muitos vendedores de SGBDs disponibilizam drivers específicos para seus sistemas.

Desta maneira, um programa cliente pode se conectar a diversos SGBDRs e enviar requisições de consultas e transações usando a API, que são processados nos servidores. Após o processamento de uma chamada de função (utilizando consultas ou programas armazenados), o resultado é fornecido pelo servidor de dados através de tabelas em memória.

Os resultados das consultas são enviados para o programa cliente, que pode processá-lo ou visualizá-lo conforme a necessidade. O conjunto resposta para uma consulta pode ser uma tabela com zero, uma ou múltiplas tuplas, dependendo de quantas linhas foram encontradas com o critério de busca.

Quando uma consulta retorna múltiplas linhas, é necessário declarar um "**CURSOR**" para processá-las. Um cursor é similar a uma variável de arquivo ou um ponteiro de arquivo, que aponta para uma única linha (tupla) do resultado da consulta.

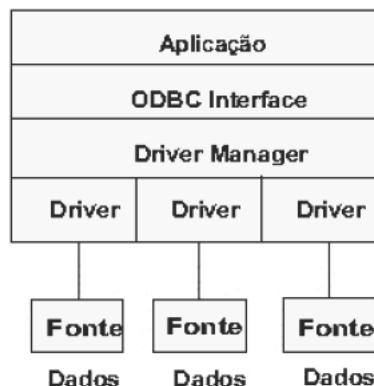
O ODBC foi desenhado para maximizar a interoperabilidade, isto é, possibilita que uma aplicação acesse diversos DBMSs ( DataBase Management Systems) usando o mesmo **código fonte**. As aplicações de base de dados chamam **funções da interface** do ODBC, que são implementadas em módulos de bancos de dados específicos chamados **drivers**.

O uso de drivers permite isolar as aplicações de chamadas específicas à base de dados da mesma forma que os drivers de impressoras isolam os programas de processamento de texto das instruções específicas das impressoras.

É importante entender que os drivers só são carregados em tempo de execução, os usuários só necessitam adicionar um novo driver cada vez que queiram acessar um novo banco de dados, ou seja, não é necessário recompilar a aplicação.

## 2.2. Arquitetura ODBC

A arquitetura do ODBC tem quatro componentes: aplicação, gerenciador de drivers, drivers e fontes de dados. A aplicação acessa o driver usando a interface ODBC. Veja a figura abaixo para entender a hierarquia dos componentes:



A **aplicação** representa os programas que chamadas as funções definidas na interface ODBC para interagir com as fontes de dados. O **gerenciador de drivers** ou *driver manager* é responsável por carregar e descarregar os drivers necessários para cada aplicação. Ele ainda processa as funções ODBC ou repassa para um dos drivers. Uma DLL (*Dynamic-Link Library*) que provê acesso aos drivers ODBC.

O **driver manager** carrega as DLL's dos drivers e direciona chamadas a funções ODBC ao driver correto. Ele também verifica algumas condições de erro e processa algumas chamadas a funções ODBC.

Os **drivers** processam as funções do ODBC, submetendo os pedidos em SQL para uma fonte de dados específica, retornando os resultados para a aplicação. Se for necessário, o driver modifica o pedido da aplicação de modo que o pedido esteja em conformidade com a sintaxe suportada pelo SGBD associado. Em outras palavras, o driver conecta uma fonte de dados, traduz comandos SQL e os submete à fonte de dados, recupera informações da fonte de dados e retorna dados para a aplicação.

Por fim **as fontes de dados**, elas são formadas pelos dados que um usuário deseja acessar, o SGBD associado, a plataforma na qual reside o SGBD, e a rede usada para acessar a plataforma. Por exemplo, uma fonte de dados ou *data source* pode ser um SGBD Oracle, rodando sobre sistema operacional OS/2, acessado por *Novell Netware*.

Apenas algumas informações importantes: (1) podem existir múltiplos drivers e fontes de dados, o que permite às aplicações acessarem simultaneamente a dados provenientes de mais do que uma fonte; (2) a API do ODBC é usada em dois lugares: entre a aplicação e o *driver manager*, e entre o *driver manager* e cada *driver* individual. A ligação entre o *driver manager* e os *drivers* é por vezes referida como sendo o SPI (*Service Provider Interface*). Para o ODBC, as APIs e os SPIs são os mesmos, ou seja, o *driver manager* e cada *driver* tem a mesma interface para as mesmas funções.

## Questões Comentadas

Vamos agora apresentar um conjunto de questões resolvida que servirão para fixação do conteúdo. Sempre que possível vamos inserir algum detalhamento teórico na explicação da questão. Esperamos que vocês gostem.



### 03. Ano: 2016 Banca: FCC Órgão: TRE-SP Cargo: Analista Judiciário de TI – Q. 57.

Durante a análise das características típicas de um Sistema Gerenciador de Bancos de Dados Relacional – SGBDR, um Analista de Sistemas verifica:

- I. Oferece suporte a várias linguagens sem a necessidade de uma sintaxe definida em quaisquer uma delas no âmbito de Data Manipulation Language – DML e Data Definition Language – DDL, porém deve ter, pelo menos, uma das linguagens com sintaxe restritiva e bem definida no âmbito de Transaction Control Language – TCL e Data Control Language – DCL.
- II. A Inserção, a atualização e a eliminação é de alto nível, ou seja, a capacidade de manipular a relação base ou relações derivadas como um operador único é aplicável não somente à recuperação de dados, mas também à inserção, alteração e eliminação de dados.
- III. Representa a descrição do Banco de Dados no nível físico na forma de dados em tabelas, permitindo que usuários autorizados apliquem formas distintas de manipular os dados nessas tabelas.

Estão corretas as características que constam APENAS em

- (A) II e III.
- (B) III.
- (C) I.
- (D) I e II.
- (E) II.

**Comentário:** Vamos comentar cada uma das alternativas acima.

Percebiam que na alternativa I ele escreve com um vocabulário um pouco rebuscado, mas diz, mais ou menos, o seguinte. Não existe uma sintaxe bem definida para os comandos de manipulação de dados e criação de objetos em SQL. Isso é um absurdo, o sucesso da linguagem passa por uma sintaxe bem definida, portanto alternativa I está **incorrecta**.

Na alternativa II o examinador diz que a existência dos comandos de INSERT, UPDATE e DELETE permite a utilização destes comandos para execução das tarefas de inserção, atualização e eliminação. Nada mais justo e correto, não acha? Sendo assim, a alternativa II está correta.

A ideia do modelo de dados relacional é justamente reduzir o conhecimento a respeito das estruturas físicas do banco de dados. Vejam que ele representa o

modelo em um nível lógico. Desta forma, a alternativa também se encontra **errada**.

Juntando os comentários acima, podemos marcar nossa resposta na alternativa E, que bate perfeitamente com o gabarito da questão.

**Gabarito: E.**



**04. Ano: 2017 Banca: IADES Órgão: Hemocentro Cargo: Analista de Tecnologia da Informação Q.**

QUESTÃO 26 - Considere as tabelas aluno (matricula INT, nome CHAR, cod\_curso INT) e curso (cod\_curso INT, curso CHAR, área CHAR), apresentadas a seguir.

matricula	nome	cod_curso
1	Joao	1
2	Pedro	1
3	Maria	2
4	Jose	2
5	Ana	3
6	Paulo	3

cod_curso	curso	area
1	Engenharia Civil	Exatas
2	Lingua Portuguesa	Humanas
3	Historia	Humanas
4	Matematica	Exatas

Qual dos comandos Structured Query Language (SQL) pode ser utilizado para listar somente os alunos que cursam História?

- (A) select aluno.Nome from aluno, curso where aluno.cod\_curso = curso.cod\_curso AND curso.curso = 'Historia';
- (B) select aluno.Nome from aluno,curso where curso.curso = 'Historia';
- (C) select aluno.Nome from aluno,curso where aluno.cod\_curso = 'Historia';
- (D) select curso.curso from aluno,curso where aluno.cod\_curso = curso.cod\_curso AND curso.curso = 'Historia';
- (E) select aluno.Nome from aluno,curso where curso.curso = 'Historia' and aluno.cod\_curso != curso.cod\_curso;

**Comentário:** Vamos analisar cada uma das alternativas acima.

Na alternativa A temos uma seleção que é feita sobre um produto cartesiano das tabelas aluno e curso. Em seguida é feita uma seleção com um predicado composto por duas partes, a primeira serve para limitar as colunas onde os atributos cod\_curso das duas tabelas são iguais, vejam que isso faz com que o produto cartesiano se transforme em uma junção. A segunda parte limita as linhas aos atributos que possuem o valor "historia" associado ao campo curso. Vejam que é exatamente isso que é solicitado no enunciado, sendo, portanto, a nossa resposta.

Agora vamos avaliar os erros da demais alternativas. Na letra B temos um produto cartesiano sem a igualdade entre os códigos do curso na cláusula WHERE. Perceba que esse fato vai fazer com que o retorno seja bem maior em termos de linhas e que os pares obtidos não possuem um relacionamento, que seria obtido com a igualdade.

Na alternativa C o erro é ainda mais grosseiro, observe que o campo aluno.cod\_curso é um atributo numérico e está sendo comparado com uma string de caracteres.

A alternativa D apresenta como resultado a coluna incorreta na cláusula select. Já a alternativa E utiliza o comparador de desigualdade na cláusula where. Em ambos os casos o resultado obtido é divergente do solicitado na questão.

**Gabarito:** A.



**05. Ano: 2017 Banca: IADES Órgão: Hemocentro Cargo: Analista de Tecnologia da Informação Q.**

QUESTÃO 28 - Considere as tabelas aluno (id INT, nome CHAR, curso INT) e curso (id INT, nome CHAR), apresentadas a seguir.

**aluno**

id	nome	curso
1	Joao	2
2	Maria	2
3	Pedro	1
4	Ana	4
5	Tiago	NULL

**curso**

id	nome
1	JAVA
2	C++
3	C#
4	PHP

Após a execução de uma consulta Structured Query Language (SQL), produziu-se exatamente o seguinte resultado:

```
+-----+-----+
| nome | nome |
+-----+-----+
| Pedro | JAVA |
| Joao  | C++  |
| Maria | C++  |
| Ana   | PHP  |
+-----+-----+
4 rows in set (0.00 sec)
```

Qual dos comandos SQL a seguir foi utilizado?

- (A) SELECT a.nome, c.nome FROM aluno a RIGHT JOIN curso c ON a.curso = c.id;
- (B) SELECT a.nome, c.nome FROM aluno a LEFT JOIN curso c ON a.curso = c.id  
UNION SELECT a.nome, c.nome FROM aluno a RIGHT JOIN curso c ON a.curso = c.id;
- (C) SELECT a.nome, c.nome FROM aluno a LEFT JOIN curso c ON a.curso = c.id;
- (D) SELECT a.nome, c.nome FROM aluno a INNER JOIN curso c ON a.curso = c.id;
- (E) SELECT a.nome, c.nome FROM aluno a RIGHT JOIN curso c ON a.curso = c.id UNION SELECT a.nome, c.nome FROM aluno a LEFT JOIN curso c ON a.curso = c.id;

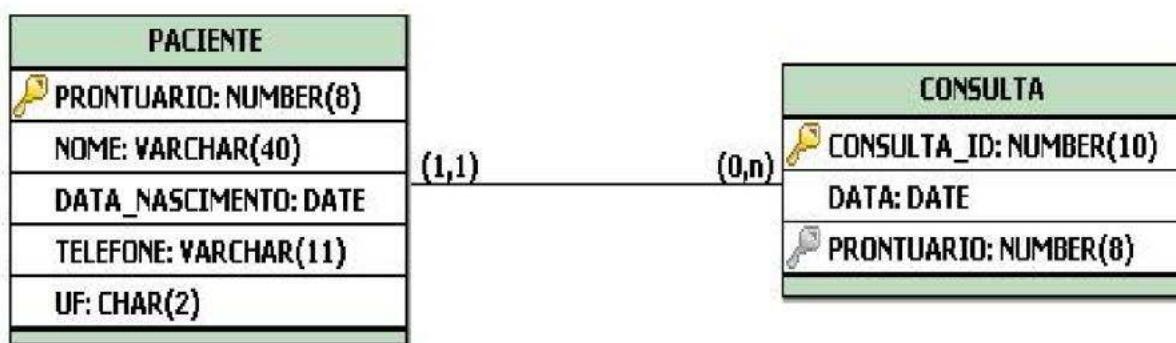
**Comentário:** Quando observamos as tabelas e os resultados podemos perceber que uma junção interna (INNER JOIN) foi executada sobre as tabelas aluno e curso para se obter o resultado esperado. Desta forma, a única alternativa que apresenta INNER JOIN no comando é a alternativa D, que é a resposta correta.

**Gabarito:** D.



#### 06. ANO: 2012 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI – BANCO DE DADOS

Considere o modelo lógico de um banco de dados, a seguir, implementado em um ambiente relacional para responder às questões 30, 31 e 32.



QUESTÃO 30 - Assinale a alternativa que apresenta, corretamente, o código que altera as datas das consultas de todos os pacientes residentes no estado de Goiás, marcadas no mês de janeiro de 2013, para o dia 04/02/2013.

(A) UPDATE CONSULTA

```
SET DATA = '04/02/2013'  
WHERE DATA > '01/01/2013' AND DATA <  
'31/01/2013'  
AND PRONTUARIO IN (SELECT PRONTUARIO  
FROM PACIENTE  
WHERE UF='GO').
```

(B) UPDATE CONSULTA

```
SET DATA = '04/02/2013'  
WHERE DATA BETWEEN '01/01/2013' AND  
'31/01/2013'  
AND PRONTUARIO IN (SELECT PRONTUARIO  
FROM PACIENTE  
WHERE UF='GO').
```

(C) UPDATE CONSULTA

```
SET DATA = '04/02/2013'  
WHERE PACIENTE.PRONTUARIO=CONSULTA.  
PRONTUARIO  
AND PACIENTE.UF='GO'  
AND DATA IN ('%01/2012').
```

(D) UPDATE CONSULTA

```
SET DATA = '04/02/2013'  
WHERE PACIENTE.PRONTUARIO=CONSULTA.  
PRONTUARIO  
AND PACIENTE.UF ='GO'  
AND DATA IN ('%JANEIRO/2012%').
```

(E) UPDATE CONSULTA

```
SET DATA = '04/02/2013'  
WHERE DATA BETWEEN '01/01/2013' AND  
'31/01/2013'  
AND PRONTUARIO = (SELECT PRONTUARIO  
FROM PACIENTE  
WHERE UF='GO').
```

**Comentários:** Vamos analisar os erros de cada uma das alternativas diferentes da resposta. Começando pela alternativa A, nela o erro acontece por não considerarmos o primeiro e o último dia do mês de janeiro, veja que o sinal de  $>$  e  $<$  são usados nas comparações.

A alternativa B é a nossa resposta, ela usa corretamente o comando `between` que inclui os valores limites da comparação, ou, em outras palavras, considera o conjunto fechado dos valores passados como limite para o intervalo.

Na alternativa C existe um erro crasso na construção do comando, não é possível referenciar a tabela PACIENTE diretamente na cláusula WHERE, a melhor forma é usar um select como descrito nas alternativas A e B.

O erro da alternativa D é semelhante ao da alternativa C, além disso, o uso do DATA IN ('%JANEIRO/2012%') não faz sentido, deveríamos ter uma sub consulta ou uma lista de valores dentro dos parênteses.

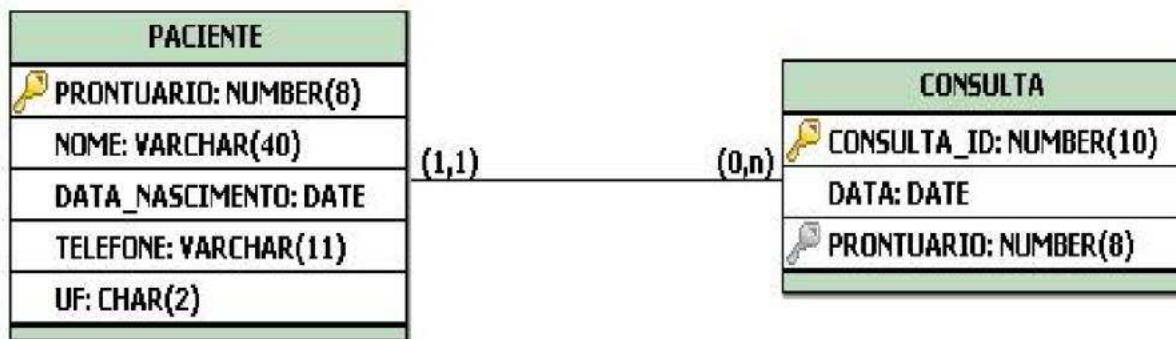
A alternativa E usa o sinal de igual para comparar um elemento a um conjunto, o que não faz sentido, poderíamos usar o IN ou EXISTS.

### **Gabarito: B**



**07. ANO: 2012 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI  
- BANCO DE DADOS**

Considere o modelo lógico de um banco de dados, a seguir, implementado em um ambiente relacional para responder às questões 30, 31 e 32.



QUESTÃO 31 Para aumentar o tamanho do campo NOME da tabela PACIENTE para 50, sem mudar o seu tipo, utiliza-se o comando:

- (A) ALTER COLUMN.
  - (B) CREATE OR REPLACE TABLE.
  - (C) DEFINE COLUMN.
  - (D) DROP TABLE.
  - (E) ALTER TABLE.

**Comentários:** Cada SGBD usa uma sintaxe específica para alterar o tamanho de uma coluna, mas todos utilizam o comando ALTER TABLE, vejam abaixo:

## **SQL Server / MS Access:**

```
ALTER TABLE table_name
```

```
ALTER COLUMN column_name datatype
```

**MySQL / Oracle (prior version 10G):**

```
ALTER TABLE table_name
```

```
MODIFY COLUMN column_name datatype
```

**Oracle 10G and later:**

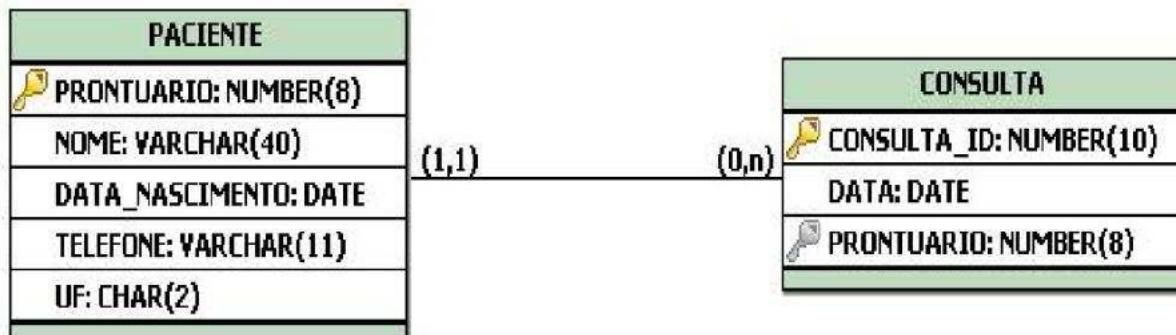
```
ALTER TABLE table_name
```

```
MODIFY column_name datatype
```

**Gabarito: E**

**08. ANO: 2012 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI  
– BANCO DE DADOS**

Considere o modelo lógico de um banco de dados, a seguir, implementado em um ambiente relacional para responder às questões 30, 31 e 32.



QUESTÃO 32 - Assinale a alternativa que apresenta, corretamente, o código que gera um relatório contendo os nomes dos pacientes em ordem alfabética e que possuem consultas marcadas para o dia 07/03/2013.

(A) 

```
SELECT P.NOME
FROM PACIENTE AS P, CONSULTA AS C
WHERE P.PRONTUARIO = C.PRONTUARIO
AND C.DATA = '07/03/2013'
ORDER BY P.NOME ASC.
```

(B) 

```
SELECT P.NOME
FROM PACIENTE AS P, CONSULTA AS C
WHERE P.PRONTUARIO = C.PRONTUARIO
AND C.DATA = '07/03/2013'
ORDER BY P.NOME DESC.
```

(C) 

```
SELECT P.NOME
```

```
FROM PACIENTE AS P, CONSULTA AS C
WHERE P.PRONTUARIO = C.PRONTUARIO
AND P.DATA_NSC = '07/03/2013'
ORDER BY P.NOME ASC.
```

(D) SELECT P.NOME  
FROM PACIENTE AS P  
WHERE P.PRONTUARIO = C.PRONTUARIO  
AND C.DATA = '07/03/2013'  
ORDER BY P.NOME ASC.  
(E) SELECT P.NOME  
FROM PACIENTE AS P, CONSULTA AS C  
WHERE P.PRONTUARIO = C.PRONTUARIO  
AND C.DATA IN ('07/03/2013')  
GROUP BY P.NOME ASC.

**Comentários:** Vamos para os comentários de cada uma das alternativas. A alternativa A é nossa resposta, vejam que ela apresenta um produto cartesiano com uma restrição de igualdade entre os valores de prontuário presentes nas duas relações. A próxima restrição refere-se à data da consulta, não confundir com a data de nascimento. Por fim, devemos ordenar a consulta em ordem alfabética.

As demais alternativas apresentam erros, a letra B ordena o resultado de forma decrescente. A alternativa C mostra a utilização da data de nascimento ao invés da data de consulta. Na letra D não foi inserida na cláusula FROM a relação consulta, desta forma ela não pode ser utilizada na comparação da cláusula WHERE o alias C, que deveria referenciar uma relação. Por fim, a alternativa E utiliza a sintaxe do IN com uma variável do tipo data, geralmente, utilizamos essa cláusula com tipos numéricos ou cadeias de caracteres.

**Gabarito: A**



#### 09. ANO: 2012 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI – BANCO DE DADOS

QUESTÃO 38 - Triggers são mecanismos úteis para alertar as pessoas ou para iniciar certas tarefas, automaticamente, quando certas condições são atendidas. A respeito deste assunto, assinale a alternativa correta.

- (A) Não se pode utilizar um trigger para a inclusão de uma nova tupla, apenas para verificação de tuplas existentes.  
(B) O modelo de trigger, conhecido como modelo evento-condição-ação, diz que para criar um mecanismo trigger, há que se cumprir dois requisitos: especificar

quando um trigger deve ser executado e as ações a serem tomadas, quando for executado.

(C) Os sistemas gerenciadores de bancos de dados não oferecem suporte interno para enviar e-mail, a partir de triggers.

(D) A cláusula `for each row`, no código do trigger, cria uma variável `nrow` chamada variável de transição.

(E) Triggers podem ser ativados antes, durante ou depois de um evento `insert`, `delete` ou `update`.

**Comentários:** Vamos relembrar a sintaxe da criação de um TRIGGER.

```
CREATE TRIGGER <trigger name>
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT | DELETE | UPDATE [ OF <column list> ] }
ON <table or view name> [ REFERENCING <alias options> ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( <search condition> ) ]
<triggered SQL statements>
```

Vejam que é possível criar um gatilho após a inclusão de uma nova linha, basta utilizar `AFTER INSERT FOR EACH ROW`. Isso invalida a alternativa A.

A alternativa B é a nossa resposta e ela nos relembra que a ideia do evento-condição-ação. Nesse contexto, a definição de uma condição para execução de um TRIGGER é opcional. O que nos leva a ter dois elementos obrigatórios o evento que vai disparar o gatilho e a ação a ser executada.

Vários sistemas de banco de dados oferecem chamadas à funções e tarefas externas como execução de um programa ou envio de um e-mail. O que inviabiliza a alternativa C.

A cláusula `FOR EACH ROW` especifica que o gatilho será executado após a inserção, deleção ou atualização de cada linha da tabela.

Triggers podem ser acionados antes, durante e ao invés de um evento de `insert`, `delete` ou `update`, como podemos observar na sintaxe do comando descrito acima.

**Gabarito: B**



#### 10. ANO: 2012 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI – BANCO DE DADOS

QUESTÃO 35 - A cláusula `where` é usada para escolher quais linhas de dados se deseja obter. Assinale a alternativa que apresenta somente condições que podem ser utilizadas nesta cláusula.

(A) `in`, `between` e `for`.

(B) `in between` e `to`.

(C) `for`, `to` e `like`.

- (D) in, between e like.  
(E) is null, and e for.

**Comentários:** Dentre as opções presentes nas alternativas acima o FOR e o TO **não** representam condições que podem ser utilizadas na cláusula WHERE, com isso invalidamos as alternativas A, B, C e E. Restando apenas a alternativa D que é a nossa resposta.

**Gabarito: D**



**11. ANO: 2014 BANCA: IADES ORGÃO: EBSERH CARGO: ANALISTA DE TI - PROCESSO**

QUESTÃO 37 - É correto afirmar que o comando utilizado para gerar uma nova tabela em uma base de dados é

- (A) DO.  
(B) GENERATE.  
(C) INSERT.  
(D) CREATE.  
(E) UPDATE.

**Comentários:** Essa questão é bem tranquila, basta termos o conhecimento do comando usado para a criação de uma nova tabela, que é o CREATE TABLE. Neste caso a resposta apresenta-se na alternativa D.

**Gabarito: D**



**12. ANO: 2013 BANCA: IADES ORGÃO: SUDAM CARGO: ANALISTA TÉCNICO ADMINISTRATIVO – ESPECIALIDADE: TECNOLOGIA DA INFORMAÇÃO**

QUESTÃO 48 - A otimização de consultas SQL envolve diversas técnicas – entre as quais, algumas universais – que formam o arsenal inicial na busca de desempenho. Entre essas técnicas estão

- I - criar índices para tabelas cujo número de registros é grande.  
II - utilizar paralelismo em consultas nas quais o hardware e o sistema gerenciador de banco de dados possuírem características de multiprocessamento.  
III - evitar o uso de recursão.  
IV - dividir uma instrução SQL complexa em diversas instruções simples que retornem o mesmo resultado final.
- A quantidade de itens certos é igual a

(A) 0. (B) 1. (C) 2. (D) 3. (E) 4.

**Comentários:** Essa é uma questão que vai além do escopo desta aula. Mas, se utilizarmos um pouco de heurística e nosso conhecimento em computação e sistemas, podemos chegar à resposta correta. De cara, posso afirmar que todas as alternativas estão corretas.

I – Indexar tabelas grandes melhora o desempenho e reduz a quantidade de acesso ao disco.

II – Paralelismo aumenta o poder de processamento de um sistema com a distribuição entre diversos processadores da execução de uma consulta ao banco de dados, por exemplo.

III – Recursão é uma operação computacional que pode estourar a pilha ou a memória devido à necessidade de guardar o contexto a cada chamada recursiva.

IV – A velha técnica de dividir para conquistar, essa ideia de dividir uma consulta complexa em várias etapas é um dos pilares do desenvolvimento do conceito de map-reduce. Do ponto de vista de SQL, o processamento de consulta distribuído pode oferecer uma divisão adequada para o processamento de uma consulta.

Vejam que todos os itens estão corretos e apresentam técnicas de melhoria do desempenho.

**Gabarito: E**



### 13. ANO: 2013 BANCA: IADES ORGÃO: METRO-DF CARGO: ANALISTA DE SISTEMAS

QUESTÃO 28 - A Linguagem de Manipulação de Dados (DML) é usada para recuperação, inclusão, exclusão e modificação de informações em bancos de dados. Ela é dividida em dois tipos: procedural e declarativa. A respeito desses dois tipos de DML, assinale a alternativa correta.

- (A) Na linguagem declarativa, o usuário não precisa especificar como os dados serão obtidos no banco de dados.
- (B) A linguagem declarativa de manipulação de dados especifica como os dados devem ser obtidos no banco de dados.
- (C) A linguagem procedural de manipulação de dados é baseada na orientação a objetos, sendo subdividida em classes.
- (D) O comando SELECT é próprio do tipo procedural de DML e é usado para definir como os dados devem ser acessados.
- (E) As DML procedurais requerem do usuário a especificação de qual dado é necessário, sem especificar como obtê-lo.

**Comentários:** SQL é uma linguagem de programação reconhecida internacionalmente e usadas para definição e manutenção de bancos de dados relacionais. A principal característica da linguagem é ser **declarativa**, ou seja,

os detalhes de implementação dos comandos são deixados para os SGBDs relacionais. Desta forma observamos nossa resposta na alternativa A.

**Gabarito: A****14. ANO: 2014 BANCA: IADES ORGÃO: TRE/PA CARGO: – ANALISTA JUDICIÁRIO: ÁREA APOIO ESPECIALIZADO – ESPECIALIDADE EM ANÁLISE DE SISTEMAS**

Questão 54 - SQL ou linguagem de consulta estruturada é uma linguagem de pesquisa declarativa usada como padrão em bancos de dados relacionais. Ela é subdividida em subconjuntos de linguagem. A respeito da linguagem de definição de dados (DDL), subconjunto da linguagem SQL, assinale a alternativa correta.

- (A) É usada para realizar inclusões, consultas, alterações e exclusões de dados.
- (B) É a linguagem que permite definir novas tabelas e elementos a ela associados.
- (C) É a linguagem que controla a autorização de acesso de usuários aos dados contidos em uma base.
- (D) Com o comando SELECT, permite-se ao usuário especificar consultas em um banco de dados.
- (E) Seu comando COMMIT finaliza uma transação dentro de um sistema de gerenciamento de banco de.

**Comentários:** Sabemos que a linguagem de definição de dados possui como principais comandos o CREATE, ALTER e DROP. Ela permite a criação de novas tabelas e outros elementos como SEQUENCE, ROLES, USER DEFINED TYPES, TRIGGERS. Desta forma, podemos encontrar a resposta na alternativa B.

A alternativa A trata da linguagem de manipulação de dados (DML). A alternativa C descreve a linguagem de controle de dados (DCL). O comando SELECT é um dos comandos DML. Por fim, a alternativa E faz referência ao COMMIT que é uma linguagem de transação de dados (DTL).

**Gabarito: B****15. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 45**

Os comandos SQL

```
create table R (a int, b int)
create table S (c int, d int)
insert into R values(1,2)
insert into R values(2,3)
```

```
insert into R values (2,3)
insert into R values (3,5)
insert into R values (4,1)
insert into S values (1,2)
insert into S values (2,1)
insert into S values (2,3)
insert into S values (3,5)
select r.a, r.b from R
where not exists
    (select * from S where s.c=r.a and s.d=r.b)
```

Produzem um resultado que, além da linha de títulos, contém:

- (A) uma linha;
- (B) duas linhas;
- (C) três linhas;
- (D) quatro linhas;
- (E) cinco linhas.

**Comentário:** Após a execução dos comandos de insert acima temos os seguintes valores nas tabelas R e S.

R (1,2) (2,3) (3,5) (4,1)

S (1,2) (2,1) (2,3) (3,5)

A consulta pede que para cada elemento do conjunto R, verificarmos se o resultado da segunda consulta é vazio, caso seja, retorne no resultado da consulta. A consulta interna verifica se o valor do campo a de R é igual ao valor do campo c de S, e se o valor do campo b de R é igual ao valor do campo d de S. Vejam que se existir essa igualdade o valor será retornado na consulta interna, o que impede que a consulta externa seja verdadeira. Em outras palavras, estamos procurando os pares de R que não estão em S, ou seja, o par (4,1).

**Gabarito:** A

**16. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 46**

O comando SQL

```
select a, sum(b) x, COUNT(*) y
from T
group by a
produz como resultado as linhas abaixo.
```

a	x	y
1	6	1
3	6	2
4	4	1
5	1	1

Na tabela T, composta por duas colunas, a e b, nessa ordem, há um registro duplicado que contém os valores:

- (A) 1 e 3
- (B) 3 e 3
- (C) 3 e 6
- (D) 4 e 2
- (E) 5 e 1

**Comentário:** Essa é uma questão interessante. Vejam que os valores mostrados na tabela é o resultado da consulta. Desta forma sabemos que a coluna x é o resultado de uma soma e que a coluna y é o resultado da contagem dos elementos com o mesmo valor, ou seja, das linhas duplicadas. Podemos perceber que o registro duplicado é o que tem valor de y maior do que 1. E que o para esse registro para a coluna a é 3, que é o mesmo valor que aparece na função de agrupamento; e b é também igual a 3 pois o valor da coluna x é a soma dos valores dos dois registros. Devemos portanto dividir o valor por 2. Assim, chegamos a nossa resposta na alternativa B.

**Gabarito:** B



**17. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 31**

Atenção: Algumas das questões seguintes fazem referência a um banco de dados relacional intitulado BOOKS, cujas tabelas e respectivas instâncias são exibidas a seguir. Essas questões referem-se às instâncias mostradas.

AUTOR

AutorID	AutorNome
1	Arthur Conan Doyle
2	Agatha Christie
3	Edgar Allan Poe

LIVRARIA

LivrariaID	LivrariaNome
1	Cultural
2	Travessia
3	Amazonas
4	Kremlin

LIVRO

LivroID	AutorID	Titulo	NumLivrarias
1	1	O Cão dos Baskervilles	NULL
2	1	As Aventuras de Sherlock Holmes	2
3	2	Assassinato no Expresso do Oriente	2
4	2	O Mistério dos Sete Relógios	3
5	3	Assassinatos na Rua Morgue	NULL

OFERTA

LivrariaID	LivroID	Preco
1	1	32
1	2	28
1	3	45
1	4	38
1	5	23
2	1	56
2	2	54
2	4	43
3	3	35
3	4	38

A tabela Livro representa livros. Cada livro tem um autor, representado na tabela Autor. A tabela Oferta representa os livros que são ofertados pelas livrarias, estas representadas pela tabela Livraria. NULL significa um campo não preenchido.

AutorID, LivrariaID e LivroID, respectivamente, constituem as chaves primárias das tabelas Autor, Livraria e Livro.

LivrariaID e LivroID constituem a chave primária da tabela Oferta.

Com relação ao banco de dados BOOKS, analise os comandos SQL exibidos a seguir:

I. select \*

```
from oferta o, livro l, autor a, livraria ll
where o.livroid=l.livroid and
o.livrariaid=ll.livrariaid and l.autorid=a.autorid
```

II. select \*

```
from oferta o inner join livro l on
o.livroid=l.livroid
inner join autor a on l.autorid=a.autorid
inner join livraria ll on
o.livrariaid=ll.livrariaid
```

III. select \*

```
from oferta o left join livro l on
o.livroid=l.livroid
left join autor a on l.autorid=a.autorid
left join livraria ll on
o.livrariaid=ll.livrariaid
```

É correto afirmar que:

- (A) somente I e II produzem resultados equivalentes;
- (B) somente I e III produzem resultados diferentes;
- (C) somente II e III produzem resultados diferentes;
- (D) todos os resultados são diferentes;
- (E) todos os resultados são equivalentes.

**Comentário:** A grande dificuldade dessa questão é observar que todas elas executam uma junção. Na primeira alternativa temos a utilização de um produto cartesiano seguido por uma seleção das colunas nas quais os valores da chave primária são iguais. Na segunda alternativa temos a utilização do **inner join** que implicitamente já realiza as duas operações da alternativa I. Por fim, temos a alternativa II, nela temos que perceber que toda oferta tem um livro associado, que por sua vez tem um autor e uma livraria. Neste encadeamento, podemos observar que não temos a possibilidade de um registro na tabela oferta que não ter correspondência na tabela livro. Desta forma, podemos observar que as três alternativas produzem resultados equivalentes.

**Gabarito:** E



**18. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 33**

No banco de dados BOOKS, o campo NumLivrarias, da tabela Livro, contém informação redundante, pois denota o número de livrarias que oferecem o livro e pode ser computado. O comando SQL que calcula e atualiza esse campo corretamente é:

- (A) update livro  
set numlivrarias=  
(select count(\*) from oferta o where  
o.livroid=livro.livroid)  
where numlivrarias=null
- (B) update livro  
set numlivrarias=  
(select count(\*) from oferta o where  
o.livroid=livro.livroid)
- (C) update numlivrarias  
from livro as  
(select count from oferta o where  
o.livroid=livro.livroid)  
where numlivrarias=null
- (D) set livro.numlivrarias=  
(select count from oferta o where  
o.livroid=livro.livroid)
- (E) set livro.numlivrarias=  
(select count(livrariaid) from oferta o where  
o.livroid=livro.livroid)  
where numlivrarias=null

**Comentário:** Precisamos entender que a informação sobre a quantidade de livrarias nas quais cada livro é vendido está presente na tabela oferta. Precisamos então contar a quantidade de linhas da tabela oferta que tenha o id do livro igual ao id da oferta. Desta forma, percebemos que nossa resposta está na alternativa B.

Como exercício, sugiro que você procure os erros das demais alternativas.

**Gabarito:** B

**19. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 34**

Quando executado no contexto do banco de dados BOOKS, o comando SQL

```
select numlivrarias from livro
where numlivrarias > 0
union
select numlivrarias from livro
```

where numlivrarias <= 0

produz um resultado cujo número de linhas, além da linha de título, é:

- (A) 1
- (B) 2
- (C) 3
- (D) 4
- (E) 5

**Comentário:** Vejam que o atributo da tabela livro denominado numlivrarias possui valores iguais a null, 2, 2, 3 e null. As consultas acima vão retornar respectivamente o conjunto (2, 2, 3) e vazio. Agora vamos aplicar a operação de união sobre esses dois conjuntos. Esta operação vai fazer com que um dos elementos 2 que está duplicado seja removido da resposta. Logo teremos apenas **duas linhas** na resposta.

**Gabarito:** B



**20. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 41**

João foi incumbido de rever um lote de consultas SQL. Como ainda é iniciante nesse assunto, João solicitou ajuda ao colega que lhe pareceu ser o mais experiente, e recebeu as seguintes recomendações gerais:

- I. use a cláusula DISTINCT somente quando estritamente necessária;
- II. dê preferência às junções externas (LEFT, RIGHT, OUTER) em relação às internas (INNER);
- III. use subconsultas escalares no comando SELECT, tais como "SELECT x,y,(SELECT ...) z ..." sempre que possível.

Sobre essas recomendações, é correto afirmar que:

- (A) nenhuma é adequada;
- (B) somente I é adequada;
- (C) somente I e II são adequadas;
- (D) somente II e III são adequadas;
- (E) todas são adequadas.

**Comentário:** Vejamos cada uma das sugestões:

I. Neste caso temos uma sugestão coerente. O uso do DISTINCT exige a execução de um algoritmo para remover registros duplicados. Para tal, é bastante possível que seja necessária alguma ordenação sobre o resultado, isso pode atrasar o processamento da consulta.

II. As junções internas são mais leves que as consultas externas, sendo processadas mais rapidamente. Portanto, não faz sentido dar preferências às junções externas.

III. Usar subconsultas com valores escalares prejudica o desempenho do SGBD pois obriga o armazenamento dos valores escalares em alguma memória temporária. Isso também deve ser evitado.

Analizando os comentários, apenas a **alternativa I está correta**.

**Gabarito:** B



**21. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: SUPORTE OPERACIONAL**  
**Questão: 42**

João escreveu a consulta SQL a seguir, executou-a corretamente e obteve um resultado contendo 100 linhas, além da linha de títulos.

```
SELECT curso, nome  
FROM aluno, curso  
WHERE aluno.codcurso = curso.codcurso  
ORDER BY curso, nome
```

As tabelas aluno e curso possuem, respectivamente, 120 e 12 linhas. No banco há ainda outras duas tabelas, pauta e disciplina, com 200 e 5 registros, respectivamente. Nessas condições, o número de linhas, além da linha de títulos, produzidas pelo comando

```
SELECT curso, nome  
FROM aluno, curso, disciplina, pauta  
WHERE aluno.codcurso = curso.codcurso  
ORDER BY curso, nome
```

seria:

- (A) 100;
- (B) 305;
- (C) 500;
- (D) 20.000;
- (E) 100.000.

**Comentário:** Vejam que a primeira informação relevante para resolvemos a questão é o fato da primeira consulta retornar 100 linhas. Em seguida, vamos aplicar a essas 100 linhas um produto cartesiano com a tabela pauta e, subsequentemente com a tabela disciplina. Sendo assim vamos ter no nosso resultado a seguinte equação:

$$100 \text{ (primeira query)} \times 200 \text{ (pauta)} \times 5 \text{ (disciplina)} = 100.000$$

Assim o nosso resultado final possui 100 mil linhas.

**Gabarito:** E



**22. BANCA: FGV ANO: 2014 ÓRGÃO: CM-RECIFE PROVA: ANALISTA LEGISLATIVO - ANALISTA DE SISTEMAS**

No SQL, o comando grant permite outorgar a um usuário (ou papel) privilégios sobre determinados recursos. Quando usado com a opção with grant option, o comando grant permite que:

- A no caso de privilégios outorgados a papéis seja possível identificar usuários que excepcionalmente não devem receber esses privilégios;
- B os privilégios outorgados não possam ser alvo de comandos revoke emitidos por usuários diferentes daquele que outorgou inicialmente;
- C os privilégios sejam outorgados em caráter temporário, sendo automaticamente removidos quando da expiração do prazo estabelecido;
- D o usuário que recebe um privilégio possa concedê-lo a outros usuários;
- E os privilégios sejam concedidos condicionalmente, e posteriormente confirmados mediante a execução automática de um procedimento de autorização.

**Comentário:** Utilizar a opção WITH GRANT OPTION indica que o usuário autorizado também poderá conceder a permissão especificada a outras entidades. Vimos isso quando tratamos de segurança em SQL.

**Gabarito D.****23. BANCA: FGV ANO: 2015 ÓRGÃO: CM CARUARU PROVA: ANALISTA LEGISLATIVO - INFORMÁTICA**

Em geral, a definição de chaves estrangeiras em bancos de dados relacionais pode vir acompanhada da especificação de procedimentos adicionais a serem adotados quando da exclusão/alteração de valores nos registros da tabela estrangeira.

```
create table R2 (
    a int not null primary key,
    b int null,
    x int not null,
    constraint FK foreign key (x) references
    R1(x)
)
```

Considerando o script acima, as opções complementares compatíveis para a definição da chave estrangeira FK são:

- A on delete cascade
- on update set null
- B on delete cascade
- on update cascade
- C on delete no action

on update set null  
 D on delete set null  
 on update restrict  
 E on delete restrict  
 on update set null

**Comentário:** A alternativa que faz mais sentido é a letra B. Não é possível utilizar a cláusula **on update set null**, pois o campo x da tabela R2 foi definido como "not null". Com os comandos da letra B, sempre que um registro de R1 tiver o valor de x alterado, essa alteração será refletida em R2. Se o registro for deletado em R1, também será excluído de R2.

**Gabarito B.**

**24. ANO: 2015 BANCA: FGV ÓRGÃO: TCE-SE PROVA: ANALISTA DE TECNOLOGIA DA INFORMAÇÃO - SEGURANÇA DA INFORMAÇÃO**

Considere duas tabelas X e Y, com as seguintes instâncias:

X		Y	
a	b	c	d
1	2	1	2
3	3	3	4
4	5	5	6
5	7	7	8
		9	1

O comando SQL que retorna

a	b	c	d
1	2	1	2
3	3	3	4
4	5	NULL	NULL
5	7	5	6
NULL	NULL	7	8
NULL	NULL	9	1

é:

- A      select \*  
from X FULL JOIN Y on X.a=Y.c
- B      select \*  
from X LEFT JOIN Y on X.a=Y.c
- C      select \*  
from Y RIGHT JOIN X on X.a=Y.c
- D      select \*  
from X CROSS JOIN Y on X.a=Y.c
- E      select \*  
from X INNER JOIN Y on X.a=Y.c

**Comentário:** Vejam que esse é um exemplo de um FULL OUTER JOIN. Ele considera todas as colunas das tabelas X e Y. Os atributos de junção são a,b em X e c,d em Y. Desta forma, quando não existe correspondente na outra relação os valores são preenchidos com NULL.

#### Gabarito A.



#### 25. Ano: 2013 Banca: FGV Órgão: SUDENE-PE Cargo: Analista Técnico Administrativo - Ciência da Computação

Com relação aos bancos de dados, os índices são uma das técnicas mais utilizadas na otimização de desempenho de consultas SQL.

A respeito dos índices, assinale V para a afirmativa verdadeira e F para a falsa.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer nos critérios de busca de uma cláusula WHERE ou HAVING.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer em uma cláusula GROUP BY e ORDER BY.

(...) Os índices provavelmente serão utilizados quando a seletividade dos dados de uma coluna indexada for baixa.

As afirmativas são, respectivamente,

- A) F, V e F.
- B) V, V e F.
- C) V, F e F.
- D) V, F e V.
- E) F, F e V

**Comentário:** Falaremos mais sobre essa questão de otimização na aula de tuning. Mas vamos comentar cada uma das alternativas acima.

I. Na primeira afirmativa temos uma demonstração consistente do uso de índices, quando os mesmos aparecem nas cláusulas WHERE e HAVING, nestes casos eles vão agilizar o processamento da consulta.

II. Vejam que se o índice for usado numa cláusula group by ou order by, por princípio, os valores estarão armazenados em posições próximas no disco rígido,

isso vai permitir uma quantidade menor de operações de I/O, que consequentemente vai melhorar a performance das operações.

III. Primeiro precisamos entender a definição de seletividade. Trata do percentual de linhas em uma coluna que possuem o mesmo valor. Uma coluna com seletividade de 5% e uma coluna com boa seletividade, por apenas 5% dos valores são repetidos. Se um índice filtra muitos os dados, dizemos que sua seletividade é alta. Do outro lado, se um índice filtra pouco os dados, podemos concluir que sua seletividade é baixa. Geralmente estamos interessados em índices de alta seletividade, de forma que essa restrição agilize a nossa consulta.

Vejam que a afirmação III vai no sentido contrário do que se espera uma seletividade baixa não ajuda, logo a alternativa está incorreta.

**Gabarito: B.**

**26. Ano: 2013 Banca: FGV Órgão: MPE-MS Cargo: Técnico - Informática**

Observe o comando SQL a seguir:

```
SELECT nome, sobrenome, PIS, anos_de_servico FROM Empregados
```

A cláusula que deve ser adicionada ao comando acima para ordenar os registros por anos de serviço, com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem, é

- A) ORDER 'anos\_de\_servico' BY ASC
- B) ORDER BY 'anos\_de\_servico'
- C) ORDER BY anos\_de\_servico DESC
- D) SORTED BY anos\_de\_servico DESC
- E) ORDER BY anos\_de\_servico ASC

**Comentário:** Percebem que quanto maior o tempo de empresa maior será a quantidade de anos de serviço. Desta forma, como a questão pede que os empregados há mais tempo na empresa apareçam primeiro devemos ordenar de forma decrescente. Outro ponto importante para responder à questão é utilizar a sintaxe correta de ordenação em SQL, qual seja, ORDER BY nome\_do\_campo DESC. Desta forma, podemos marcar o gabarito correto na alternativa C.

**Gabarito: C.**

**27. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas**

A necessidade de construir consultas aplicadas a sistemas de apoio à decisão levou à introdução de algumas construções especiais na linguagem SQL, que facilitam e estendem a agregação de dados. Dentre essas estão:

- A) oltp e olap;
- B) intersection e minus;

- C) rank e decode;
- D) cube e rollup;
- E) slicing e dicing.

**Comentário:** Em SQL, podemos utilizar operadores ROLLUP e CUBE na instrução SELECT para totalizações. O operador ROLLUP serve para mostrar que podemos obter um total geral, permitindo agregações sobre grupos de linhas. O ROLLUP não precisa que se especifique nenhum campo e ele age em conjunto com o GROUP BY e a função de agregação.

A clausula CUBE é uma extensão de ROLLUP que permite as agregações totais por todas as colunas envolvidas. Na prática cria um cubo. O exemplo abaixo determina a soma de salários usando GROUP BY CUBE (JOB, DEPTNO):

select job, deptno, sum(sal) from emp group by cube(job, deptno) order by job, deptno;		
JOB	DEPTNO	SUM(SAL)
ANALYST	20	6000
ANALYST		6000
CLERK	10	1300
CLERK	20	1900
CLERK	30	950
CLERK		4150
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
MANAGER		8275
PRESIDENT	10	5000
PRESIDENT		5000
SALESMAN	30	5600
SALESMAN		5600
	10	8750
	20	10875
	30	9400
		29025

18 rows selected

Vejam que a opção que apresenta as duas instruções de agregação descritas por SQL é a presente na alternativa D.

**Gabarito: D.**



**28. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas**

Considere as seguintes tabelas relacionais e respectivas instâncias.

R	
A	B
1	3
2	3
3	4
5	5
5	7
8	8

S	
C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

```
SELECT A,
      (SELECT COUNT (*)
        FROM S WHERE NOT R.B = S.C OR R.A = S.D
      ) X FROM R
ORDER BY A;
```

Os números que aparecem na coluna X do resultado da execução desse comando, de cima para baixo, são:

- A) 0, 0, 1, 0, 0, 0.
- B) 0, 0, 0, 0, 0, 0.
- C) 2, 2, 5, 5, 5, 5.
- D) 2, 2, 4, 5, 5, 5.
- E) 3, 3, 1, 0, 0, 0.

**Comentário:** A questão pede para que seja contabilizada a quantidade de registros que atendem a seguinte caraterísticas: **NOT R.B = S.C OR R.A = S.D**.

Vejamos o caso prático para a primeira linha da tabela R (1,3), para cada registro de S vamos verificar os dois predicados, e proceder uma operação lógica de OR entre os dois.

Na primeira linha de S temos (1, 9), vejam que R.B(3) != S.C(1), isso torna o primeiro predicado verdadeiro; e R.A(1) != S.D (9), que torna o segundo predicado falso. Então fazemos um OR entre os dois. Neste caso podemos contabilizar 1 no somatório.

Fazemos o mesmo para todas as linhas da tabela S e obtemos o resultado 2 para a primeira coluna X do resultado. Após aplicar essa lógica para todas as linhas podemos encontrar os seguintes valores para X (2, 2, 5, 5, 5, 5).

**Gabarito: C**



#### **29. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas**

Considere as seguintes tabelas relacionais e respectivas instâncias.

R	
A	B
1	3
2	3
3	4
5	5
5	7
8	8

S	
C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

```
DELETE FROM S
WHERE NOT EXISTS
    (SELECT * FROM R
     WHERE R.A = S.C AND R.B = S.D)
```

O número de registros deletados por esse comando é:

- A) 0;
- B) 1;
- C) 2;
- D) 4;
- E) 5.

**Comentário:** Vejam que essa questão trata de uma consulta correlacionada. Vamos excluir o registro de S toda vez que o conjunto de linhas da consulta interna for vazia. De forma simples, vamos apagar as tuplas de S que não tenham o valor em R. No caso concreto, a primeira linha de S será removida.

**Gabarito: B.**



### **30. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas**

Considere as seguintes tabelas relacionais e respectivas instâncias.

R	
A	B
1	3
2	3
3	4
5	5
5	7
8	8

S	
C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

```
SELECT * FROM R UNION SELECT * FROM S
```

O número de linhas produzidas por esse comando, excetuada a linha de títulos de colunas, é:

- A) 2;
- B) 5;

- C) 6;
- D) 7;
- E) 11.

**Comentário:** Questão interessante para lembrarmos que as operações com conjuntos elas vão eliminar as tuplas duplicadas. Desta forma, o nosso resultado terá apenas 7 linhas. Para a operação com conjuntos não eliminarem as tuplas duplicadas precisamos utilizar a cláusula ALL: UNION ALL, por exemplo.

**Gabarito:** D.



**31. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados**

Observe o comando SQL a seguir.

```
UPDATE X SET Y = 'Z'
```

Para que esse comando esteja corretamente formulado, quando analisado isoladamente, pressupõe-se que:

- A) Y seja uma coluna da tabela X;
- B) X seja uma coluna da tabela Y;
- C) X e Y sejam tabelas;
- D) X seja um banco de dados e Y seja uma tabela;
- E) Y seja uma coluna da tabela X e Z seja o nome de um tipo de dados válido.

**Comentário:** Veja que esta é uma questão relativamente simples. Ela procura medir nosso conhecimento a respeito da sintaxe do comando UPDATE. Observe que X deve ser o nome de uma tabela e Y um atributo desta tabela, pela forma como o argumento é passado podemos supor que é um atributo textual.

**Gabarito:** A.



**32. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados**

Analise o comando SQL a seguir.

```
SELECT DISTINCT 1 FROM X
```

Sabendo-se que a instância da tabela X não é vazia, conclui-se que a execução desse comando produz um resultado com:

- A) apenas uma linha;
- B) um conjunto de linhas contendo o valor NULL;
- C) um conjunto de linhas contendo todos os atributos de X;

- D) um número de linhas igual ao número de diferentes valores do primeiro atributo de X;  
 E) um número de linhas igual ao número de registros de X.

**Comentário:** Essa questão testa o nosso conhecimento prático sobre SQL. Você saber o que acontece se executarmos o comando: `SELECT *, 1 FROM TabelaX;`?

Basicamente o banco de dados vai te retornar todas as linhas e colunas da TabelaX e adicionar uma nova coluna com nome 1 e todos os valores iguais a 1. Você pode testar isso na prática [aqui](#). Agora que você já sabe que todos os valores da coluna 1 são iguais, o que acontece se utilizarmos a cláusula `distinct`? Justamente! Teremos como resultado apenas **uma linha cujo valor é 1**.

**Gabarito: A.**



**33. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação**

X		Y	
a	b	c	d
1	2	1	2
3	3	3	4
4	5	5	6
5	7	7	8
		9	1

```
delete from y
where y.c in
(select a from x union select c from y)
```

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que o número de registros removidos da tabela Y pela execução desse comando é:

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5

**Comentário:** Vejam que a questão vai fazer uma deleção de todas as linhas da tabela Y. Vamos começar verificando o resultado retornado pela consulta interna. Ela, basicamente, junta todos os valores de a em X com os de c em Y, resultando em (1, 3, 4, 5, 7, 9). Depois vamos deletar todas das linhas de Y cujos valores de c estejam nesta lista. Ou seja, apagaremos todas as **cinco** linhas de Y.

**Gabarito: E.**


**34. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação**

X		Y	
a	b	c	d
1	2	1	2
3	3	3	4
4	5	5	6
5	7	7	8
		9	1

**SELECT X.a FROM X  
WHERE NOT EXISTS  
(SELECT \* FROM Y WHERE Y.c = X.a + 1)**

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz um resultado com uma única coluna contendo somente o (s) valor(es):

- A) 4
- B) 3, 4
- C) 1, 3, 5
- D) 3, 4, 5
- E) 1, 3, 4, 5

**Comentário:** A questão permite observarmos o processo de avaliação de uma consulta correlacionada. Vamos utilizar dois cursores imaginários para analisar a questão, um cursor C1 que aponta para a primeira linha de X e um cursor C2 que aponta para a primeira linha de Y. No início do processo vamos verificar qual o valor atual de C1, no caso (1, 2). De posse desse valor vamos verificar se não existe algum elemento na consulta interna. Percebem que a consulta interna vai percorrer todas as tuplas de Y e verificar se existe algum elemento cujo valor de c é igual a x.a+1, neste caso, como x.a é igual a 1, x.a+1 ficaria igual a 2. Vejam que não existe nenhum tupla cujo valor de y.c é igual a 2. Logo, o resultado da consulta interna seria vazio e o valor atual do ponteiro C1 seria retornado na resposta.

Esse passo-a-passo seria feito para cada um dos valores de X, no final obteríamos 1, 3, 5 como valores. Essa é nossa resposta, presente na alternativa C.

**Gabarito: C.**



**35. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação**

X		Y	
a	b	c	d
1	2	1	2
3	3	3	4
4	5	5	6
5	7	7	8
		9	1

```
SELECT *
FROM X LEFT JOIN Y ON X.a = Y.c
ORDER BY X.a
```

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz:

- A) 1, 2, 1, 2  
3, 3, 3, 4  
5, 7, 5, 6
- B) 1, 2, 1, 2  
3, 4, 3, 3  
NULL, NULL, 4, 5  
5, 6, 5, 7
- C) 1, 2, 1, 2  
3, 3, 3, 4  
NULL, NULL, NULL, NULL  
5, 7, 5, 6
- D) 1, 2, 1, 2  
3, 3, 3, 4  
4, 5, NULL, NULL  
5, 7, 5, 6
- E) 1, 2  
3, 3  
4, 5  
5, 7

**Comentário:** Vejam que a questão pede para executarmos um LEFT JOIN utilizando como referências os atributos x.a e y.c. Sabemos que o LEFT JOIN considera toda as tuplas da esquerda completando com NULL todas as vezes que o atributo de junção não tiver correspondência na tabela da direita. Desta forma, conseguimos observar que a tupla que não tem correspondência seria a terceira tupla de X onde o valor de a é igual a 4, valor que não existe na coluna c de Y. Neste caso, completaremos a tupla do resultado com valor nulo. Para os demais valores de X.a temos um valor correspondente de Y.

Despois da execução da junção é solicitado que o resultado seja ordenado por x.a. Como a tabela X já está ordenada pela coluna a, podemos observar o resultado da consulta na alternativa D.

**Gabarito: D.**



**36. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação**

Views criadas nos bancos podem, de acordo com alguns critérios, ser naturalmente atualizáveis, o que significa, por exemplo, que podem ser objeto de comandos update do SQL sem a necessidade de mecanismos auxiliares ou triggers. Essa característica depende da expressão SQL que define a view e das tabelas/views de origem.

Considere alguns tipos de construções SQL que podem ser empregadas na definição de uma coluna de uma view:

- I. funções de agregação, tais como sum, avg
- II. funções escalares, tais como sin, trim
- III. expressões aritméticas
- IV. expressões condicionais, tais como case
- V. literais
- VI. subconsultas

Está correto concluir que uma determinada coluna **NÃO** pode ser objeto de atualização quando resultar de qualquer dos tipos:

- A) apresentados, exceto I, II e III;
- B) apresentados, exceto III e IV;
- C) apresentados, exceto V;
- D) apresentados, exceto VI;
- E) apresentados.

**Comentário:** Sabemos que uma visão é uma tabela cujas linhas não são armazenadas explicitamente no banco de dados, mas são calculadas conforme necessário, com base em uma definição de visão. A motivação por trás do mecanismo de visões é personalizar a maneira como os usuários veem os dados.

O padrão SQL-92 permite que atualizações sejam especificadas apenas em visões definidas em uma única tabela base, usando apenas seleção e projeção, sem nenhum uso de operação de agregação. Tais visões são denominadas de **visões atualizáveis**. O padrão SQL-1999 ampliou a possibilidade de atualização, intuitivamente, podemos atualizar um campo de uma visão, se ele é obtido de exatamente uma das tabelas bases e a chave primária dessas tabelas estiver incluída na visão.

Também temos restrições de que o operador DISTINCT não pode ser usado em definições de visões atualizáveis.

Para finalizar, o Silberschatz diz que uma alteração por meio de visão somente é permitida se a visão em questão é definida em termos de uma **relação real do banco de dados**. Observem que nenhuma das alternativas apresenta valores armazenados no banco de dados, desta forma a letra E traz a nossa resposta.

**Gabarito: E.**



**37. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação**

Considere a tabela relacional criada pelo comando

create table xx

(a int null, b int null, c int null)

Depois de instanciada com um conjunto de registros, os seguintes comandos foram executados:

select count (\*) from XX

select count (distinct A) from XX

select count (distinct B) from XX

select count (\*) from XX where C>10

select count (\*) from XX where not C>10

Sabendo-se que esses comandos produziram como resultado, respectivamente, os números 10, 10, 0, 0 e 5, analise as quatro alternativas para a definição da tabela XX:

I. CREATE TABLE XX(

    A int NULL,

    B int NULL,

    C int NULL )

II.CREATE TABLE XX(

    A int primary key,

    B int NULL,

    C int NULL )

III.CREATE TABLE XX (

    A int NULL,

    B int NULL,

    C int)

IV. CREATE TABLE XX (

    A int,

    B int primary key,

    C int NULL)

A lista com todos os comandos que são válidos e compatíveis com a instância corrente da tabela é:

- A) I, II;
- B) I, II, III;
- C) II, IV;
- D) I, III;
- E) IV.

**Comentário:** As conclusões que podemos tirar a partir do resultado das consultas no ajudam a responder à questão. A primeira consulta demonstra que a tabela tem 10 registros. A segunda consulta mostra que a coluna A possui 10 valores distintos e diferentes de NULL. A terceira apresenta a coluna B com todos os seus valores iguais a NULL, essa conclusão pode ser entendida pelo fato da cláusula DISTINCT desconsiderar valores nulos na sua contabilidade. Sendo assim, se temos o valor zero como resposta da consulta é porque todos os valores da coluna B são iguais a NULO.

As outras duas consultas devem ser usadas em conjunto. Se  $C > 10$  e  $C \leq 0$  (é a mesma coisa de dizer que  $\text{not } c > 0$ ) conseguem em conjunto contar apenas 5 tuplas é porque os demais valores de C são iguais a NULL.

Partido dessas considerações podemos construir as tabelas das alternativas I, II e III e inserir os mesmos valores sem problemas. Já a alternativa IV apresenta um erro ao definir B como chave primária, o que não é possível dado os valores nulos para o campo.

**Gabarito: B**



**38. Ano: 2014 Banca: FGV Órgão: SUSAM Cargo: Técnico de Nível Superior A - Analista de Sistemas**

A figura a seguir apresenta o diagrama das tabelas TBNotaFiscal e TBItemNotaFiscal. As tabelas 1 e 2 apresentam, respectivamente, os registros TBNotaFiscal e TBItemNotaFiscal.

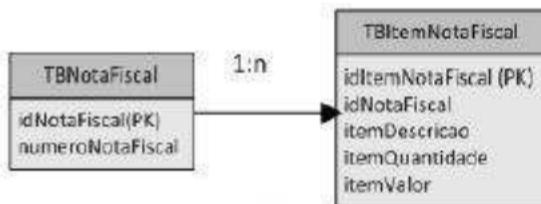


Tabela 1

TBNotaFiscal	
idNotaFiscal	numeroNotaFiscal
1	NF 0001
2	NF 0002
3	NF 0003

Tabela 2

TBItemNotaFiscal				
idItemNotaFiscal	idNotaFiscal	itemDescricao	itemQuantidade	itemValor
1	1	porca	100	0,10
2	1	parafuso	50	0,20
3	2	areia	10	10,50
4	2	cimento	15	20,00
5	2	telha	100	2,50

Os campos idCliente e idCompras são chaves primárias das tabelas TBNotaFiscal e TBItemNotaFiscal, respectivamente.

Assinale a opção que indica resposta correta para o seguinte comando SQL

```

SELECT a.numeroNotaFiscal, b.itemDescricao,
       b.itemValor, b.itemQuant, b.itemValorAS
       ValorUnitario,(b.itemValor)*(b.itemQuant)AS
       ValorTotal
FROM TBNotaFiscal a, TBItemNotaFiscal b
WHERE a.idNotaFiscal = b.idNotaFiscal
  
```

A) NF 0001      porca      0,1      100      0,1      10  
 NF 0001      parafuso      0,2      50      0,2      10

NF 0002      areia      10,5      10      10,5      105

NF 0002      cimento      20      15      20      300

NF 0002      telha      2,5      100      2,5      250

B) NF 0001      porca      0,1      100      0,1      10  
 NF 0001      parafuso      0,2      50      0,2      10  
 NF 0002      areia      10,5      10      10,5      105  
 NF 0002      cimento      20      15      20      300  
 NF 0002      telha      2,5      100      2,5      250  
 NF 0003      NULL      NULL      NULL      NULL      NULL

C)	NF 0001	porca	0,1	100	0,1	10
	NF 0001	parafuso	0,2	50	0,2	10
	NF 0002	areia	10,5	10	10,5	105
	NF 0002	cimento	20	15	20	300
	NF 0002	telha	2,5	100	2,5	250
	NF 0003	NULL	0	0	0	0
D)	NF 0001	NULL	NULL	NULL	NULL	20
	NF 0002	NULL	NULL	NULL	NULL	655
	NF 0003	NULL	NULL	NULL	NULL	0
E)	NF 0001	NULL	NULL	NULL	NULL	20
	NF 0002	NULL	NULL	NULL	NULL	655

**Comentário:** Essas alternativas ficaram um pouco difícil de visualizar, mas percebam que a nota ficas é composta de NF mais um número identificador. Agora vamos voltar nossa atenção para o comando descrito no enunciado, o SELECT lista alguns atributos presentes nas duas tabelas, em seguida faz um produto cartesiano seguido por uma seleção. Veja que estamos fazendo uma junção onde o atributo de junção é o idNotaFiscal. Este fato nos leva a percepção de que não haverá valores nulo na relação resultante da consulta. A ideia é concatenar o número da nota fiscal aos itens e suas descrições. Sendo assim, podemos observar nossa resposta na alternativa A.

**Gabarito: A.**



**39. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação**

Considere que as instâncias das tabelas T1, T2 e T3 têm, respectivamente, 1.000, 10.000 e 100.000 registros. O comando SQL

```
select 1 from t1
union
select 2 from t2
union
select 3 from t3
```

produz um resultado com:

- A) 3 linhas;
- B) 1.000 linhas;
- C) 10.000 linhas;
- D) 100.000 linhas;
- E) 111.000 linhas.

**Comentário:** Vaja que comentamos essa característica do uso de valores numéricos na cláusula SELECT. Para cada um dos select teremos uma coluna com a quantidade de linhas iguais a quantidade de linha de cada tabela e os valores para cada linha iguais ao número, 1, 2 e 3, respectivamente. Agora, o pulo do gato está na operação UNION que vai remover os valores duplicados. Sendo assim nosso resultado final será:

- (1)
- (2)
- (3)

Ou seja, apenas 3 linhas serão retornadas.

**Gabarito: A**



**40. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação**

Analise os comandos SQL a seguir, que produzem os resultados R1, R2 e R3, respectivamente.

- I. select distinct x.\* from x, y where x.a <> y.a
- II. select distinct x.\* from x  
where x.a not in (select a from y)
- III.select distinct x.\* from x  
where not exists  
(select \* from y where y.a=x.a)

Sabendo-se que nenhuma das instâncias das tabelas "x" e "y" é vazia, é correto concluir que:

- A) R1 e R2 são iguais entre si e diferentes de R3;
- B) R1 e R3 são iguais entre si e diferentes de R2;
- C) R2 e R3 são iguais entre si e diferentes de R1;
- D) R1, R2 e R3 são todos iguais entre si;
- E) R1, R2 e R3 são todos diferentes entre si.

**Comentário:** A diferença é que na opção 1 haverá um produto cartesiano entre x e y que será então filtrado na cláusula WHERE. Nas outras opções, não haverá esse produto cartesiano. Apenas comparações entre os valores existentes nas relações x e y. Vejam uma solução para a questão executada no MySQL:

```
mysql> select * from x;
```

```
+---+
| a |
+---+
| 1 |
| 2 |
```

```
| 3 |
+---+
3 rows in set (0.00 sec)
```

```
mysql> select * from y;
+---+
| a |
+---+
| 2 |
| 3 |
| 4 |
+---+
3 rows in set (0.00 sec)
```

R1

```
mysql> select distinct x.* from x, y where x.a != y.a ; // <> é o mesmo que !=
+---+
| a |
+---+
| 1 |
| 3 |
| 2 |
+---+
3 rows in set (0.00 sec)
```

R2

```
mysql> select distinct x.* from x
    -> where x.a not in (select a from y) ;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

R3

```
mysql> select distinct x.* from x
    -> where not exists
    -> (select * from y where y.a=x.a) ;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

Analizando os comandos acima podemos perceber que R2 e R3 produzem o mesmo resultado. Já R1 possui um resultado distinto. A alternativa C contém tal informação, sendo, portanto, nossa resposta.

**Gabarito: C.**

**41. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação**

Analise as instâncias das tabelas R1 e R2 e o comando SQL, mostrados a seguir.

R1
x
1
3
4
5
6
7
8

R2	
a	b
2	0
3	0
7	0
9	0
10	0

```
update R2
set b=(select count(x) from R1 where x > a)
```

Após a execução do comando, o conteúdo da coluna "b" da tabela R2 passa a ser, de cima para baixo:

- A) 5, 5, 5, 5;
- B) NULL, NULL, NULL, NULL, NULL;
- C) 5, 3, 1, 0, 0;
- D) 0, 0, 0, 0, 0;
- E) 5, 3, 1, NULL, NULL.

**Comentário:** Para resolver a questão precisamos perceber que o valor de b será atualizado com a quantidade de números presentes na coluna x que sejam maiores que a. Por exemplo, quantos valores da coluna x são maiores que 2? São 5 valores, no caso, (3, 4, 6, 7, 8). Fazendo esse exercício para as demais linhas da tabela R2 temos os seguintes valores da coluna b (5, 3, 1, 0, 0). Tais valores estão presentes na alternativa C.

**Gabarito: C.**

**42. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas**

Analise o comando de criação de uma tabela relacional mostrado a seguir.

```
create table inscrição
(matrícula int not null,
disciplina int not null,
nota float null,
constraint pk_inscrição primary key (matrícula,
disciplina),
constraint fk_inscrição_aluno
foreign key (matrícula) references aluno
(matrícula)
)
```

Sabendo-se que a coluna matrícula constitui a chave primária da tabela aluno, está correto concluir que

- A) aluno e inscrição têm um relacionamento 1:1 entre si.
- B) aluno e inscrição têm um relacionamento 1:n entre si.
- C) aluno e inscrição têm um relacionamento n:1 entre si.
- D) aluno e inscrição têm um relacionamento m:n entre si.
- E) inscrição é uma especialização de aluno.

**Comentário:** Essa questão nos lembra do assunto que vimos anteriormente quando tratamos de modelagem conceitual. Vejam que cada aluno pode ser matriculado em várias disciplinas, cada disciplina por meio de uma inscrição, mas percebam que cada inscrição só pode ter um aluno. Sendo assim, o relacionamento entre aluno e inscrição é 1:n. Temos, portanto, nossa resposta na alternativa B.

**Gabarito: B**



**43. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas**

Analise o comando SQL a seguir.

```
select x,
       sum(nota) as soma,
       count(nota) as numero
  from inscrição
 group by x
 having ???
```

Assinale a opção que indica a expressão que, ao ser utilizada para substituir o trecho "???", **INVALIDA** o comando SQL acima.

- A) count (nota) > 1
- B) x=2
- C) max (nota) = 10
- D) nota > 7
- E) (select max(nota) from inscrição) > 5

**Comentário:** Pela lógica a restrição sobre o atributo x deveria fazer parte da cláusula WHERE. Do ponto de vista de performance é melhor você restringir as linhas para depois fazer os agrupamentos e em seguida as restrições sobre o agrupamento. Contudo não é bem assim, como pode ser visto [link](#).

O HAVING pode ser usado em funções agregadas ou AGRUPADAS. Assim, como o X que está no GROUP BY, também pode ser usado na cláusula HAVING. No caso da questão, o HAVING pode fazer restrições usando as funções de agregação e sobre os atributos que estão sendo AGRUPADOS pelo GROUP BY.

Observem que o atributo nota não aparece no group by, portanto, não pode ser usado isoladamente.

**Gabarito: D.**



#### 44. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Na maioria das implementações SQL, pode-se considerar que as expressões lógicas possam assumir três valores, verdadeiro (T), falso (F) e desconhecido (?). Isso decorre principalmente da manipulação de valores nulos (NULL).

Assim sendo, analise as quatro expressões lógicas a seguir.

not ?

F or ?

T and ?

? or T

Assinale a opção que apresenta os valores finais das expressões lógicas acima, na ordem de cima para baixo.

A) F; ?; T; T

B) F; F; T; T

C) ?; ?; ?; ?

D) ?; ?; ?; T

E) ?; F; ?; ?

**Comentário:** A última questão trata da lógica de três valores usada em SQL. Vejam que **not ?** é igual a ?. **F or ?** é igual ?. **T and ?** é igual a ?. E ? or T é igual a T. Neste último caso basta substituir a ? por T e por F e verá que teremos os mesmos resultados. Sendo assim, podemos concluir que ? or T é igual a T.

Analisando todas as alternativas podemos encontrar nossa resposta na letra D.

**Gabarito: D**



#### 45. BANCA: ESAF ANO: 2016 ÓRGÃO: ANAC CARGO: ANALISTA DE TI - QUESTÃO 34

Em SQL, algumas consultas precisam de que os valores existentes no banco de dados sejam buscados e depois usados em uma condição de comparação. Elas podem ser formuladas por meio de consultas.

- a) concorrentes.
- b) comparadas.
- c) segmentadas.
- d) hierarquizadas.
- e) aninhadas.

**Comentário:** Uma consulta SQL é aninhada quando ela está dentro de outra consulta SQL. A consulta aninhada normalmente aparece como parte de uma condição nas cláusulas WHERE ou HAVING. As consultas aninhadas também podem ser utilizadas na cláusula FROM.

As consultas aninhadas podem ser utilizadas como um procedimento em que a consulta é executada apenas uma vez, conhecida como não correlacionada. Outra opção é a consulta ser executada repetidas vezes, neste caso falamos que ela é correlacionada.

**Gabarito: E****46. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 63 E 64**

Acerca de linguagens de definição e manipulação de dados, julgue os itens subsecutivos.

63 Apelido ou column alias não pode ser utilizado na cláusula WHERE.

64 Em uma coluna definida como NUMBER (7,2), o valor 34567.2255 será armazenado como 34567.23.

**Comentários:** Vamos comentar cada uma das alternativas acima.

A primeira, refere-se ao alias, que pode ser utilizado implicitamente ou representado pela palavra-chave AS. Pode ser usado sobre colunas na cláusula SELECT ou em tabelas na cláusula FROM. Contudo, não dever ser usado na cláusula WHERE, geralmente referenciamos o alias definido anteriormente.

A segunda alternativa trata da sintaxe de definição da variável NUMBER, o primeiro número refere-se à quantidade total de algarismos do número, já o segundo valor entre parênteses diz respeito a quantidade de casas decimais. Observem que a questão se encontra correta.

**Gabarito: C C****47. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 65 A 66**

Julgue os próximos itens, relativos a SQL.

65 O comando SQL ilustrado a seguir atualiza os dados dos empregados do departamento (id\_departamento) 50 que têm como função (id\_funcao) VENDEDOR para o departamento 80 e gerente (id\_gerente) 145.

```
UPDATE empregados
SET id_departamento = 80,
id_gerente = 145
WHERE id_departamento = 50
AND funcao = 'VENDEDOR';
```

66 O comando SQL mostrado a seguir fará uma consulta na tabela empregados e retornará os campos primeiro\_nome, sobrenome e salario de todos os empregados do departamento (id\_departamento) 40, ordenados pelo campo sobrenome.

```
SELECT primeiro_nome, sobrenome, salario
FROM empregados
WHERE id_departamento = 40
ORDER BY sobrenome
```

**Comentários:** Vamos comentar as alternativas acima.

Na alternativa 65 o CESPE cometeu um erro proposital na digitação do atributo função (id\_funcao), percebam que não temos o **id** na descrição do comando. A ideia é a seguinte. Pense que a tabela empregados terá um identificador para departamento e função. Esses podem ter suas descrições e outros atributos descritos em suas tabelas. Desta forma, na tabela empregados teríamos apenas o identificador. Alternativa errada.

A questão 66 está perfeitamente correta! Ela apresenta a construção de um comando select com a sintaxe e as cláusulas descritas na ordem certa.

**Gabarito: E C**



#### 48. BANCA: CESPE ANO: 2016 ÓRGÃO: TCE-SC CARGO: AUDITOR DE TI

Com relação aos bancos de dados relacionais, julgue os próximos itens.

94 O catálogo de um sistema de gerenciamento de banco de dados relacional armazena a descrição da estrutura do banco de dados e contém informações a respeito de cada arquivo, do tipo e formato de armazenamento de cada item de dado e das restrições relativas aos dados.

95 Denomina-se visão uma tabela única derivada de uma ou mais tabelas básicas do banco. Essa tabela existe em forma física e viabiliza operações ilimitadas de atualização e consulta.

96 Em bancos de dados relacionais, as tabelas que compartilham um elemento de dado em comum podem ser combinadas para apresentar dados solicitados pelos usuários.

**Comentário:** O dicionário de dados ou catálogo de dados contém as descrições das estruturas dos objetos presentes na base de dados. Presente em todos os

SGBDs relacionais ele guarda os metadados ou informações a respeitos dos objetos armazenados. Podemos marcar como correta a assertiva 94.

A definição de visão presente no padrão SQL/ANSI é de uma estrutura temporária que armazena informações advinda de uma ou mais tabelas. A visão não é armazenada fisicamente em disco e é removida ou apagada ao final da sua utilização. Sendo assim, a alternativa 95 encontra-se incorreta.

Dentro do contexto de bancos de dados relacionais, é possível usar as operações de junção. Essas operações utilizam atributos que operam sobre o mesmo domínio presentes em cada uma das tabelas. Esses atributos são utilizados para juntar ou relacionar uma tabela com a outra, sempre que tivermos os mesmos valores em ambas as tabelas. Vejam que temos mais uma vez uma alternativa correta.

**Gabarito: C E C**



**49. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI  
-QUESTÃO 13**

Acerca de SQL (structured query language), assinale a opção correta.

A A otimização semântica de consultas utiliza restrições existentes no banco de dados (como atributos únicos, por exemplo) com o objetivo de transformar um SELECT em outro mais eficiente para ser executado.

B Quando os registros de uma tabela estão ordenados fisicamente em um arquivo, segundo um campo que também é campo-chave, o índice primário passa a se chamar índice cluster.

C Em uma mesma base de dados, uma instrução de SELECT com união de duas tabelas (INNER JOIN) e uma instrução de SELECT em uma tabela utilizando um SELECT interno de outra tabela (subquery) produzem o mesmo resultado e são executadas com o mesmo desempenho ou velocidade.

D A normalização de dados é uma forma de otimizar consultas SQL, ao apresentar um modelo de dados com um mínimo de redundância. Isso é atingido quando o modelo estiver na quinta forma normal (5FN).

E Para obter a quantidade de linhas que atendem a determinada instrução SQL, o processo mais eficiente e rápido é executar o comando SELECT e aplicar uma estrutura de loop para contar as linhas resultantes.

**Comentário:** Vamos então analisar cada uma das alternativas acima.

A Uma transformação que é válida somente porque certa restrição de integridade está em efeito é chamada **transformação semântica** e a otimização resultante é chamada **otimização semântica**. A otimização semântica pode ser definida como o processo de transformar uma consulta especificada em outra consulta, qualitativamente diferente, mas da qual se garante que produzirá o mesmo resultado que a original, graças ao fato de que os dados com certeza satisfazem a uma determinada restrição de integridade. Vejam que a definição está de acordo com o descrito na alternativa, logo essa é a nossa resposta.

B Quando tratamos de índices podemos de forma resumida classificá-los em três categorias: **Índice primário**, baseado na chave de ordenação; **Índice de agrupamento (clustering)**, baseado no campo de ordenação não-chave de um arquivo e **Índice secundário**, baseado em qualquer campo não ordenado de um arquivo. Observem que a alternativa tenta confundir a definição de índice primário com índice de cluster.

C Não existe nenhuma garantia para saber qual das duas instruções será executada de forma mais rápida, vai depender, por exemplo, do perfil dos dados, do algoritmo utilizado pelo SGBD na execução das consultas e da forma como os índices são criados para cada uma delas. Desta forma, a alternativa está **incorrecta**.

D A normalização é uma atividade que reduza a redundância dos dados dentro do banco de dados e as anomalias de atualização. Não existe um compromisso do processo de normalização com a otimização de consultas. Se lembramos da estrutura dos modelos dimensionais, sabemos que eles são desnormalizados por uma questão de performance.

E Mais uma vez, a velocidade da consulta ou seu desempenho vai depender da forma como os dados e os índices estão estruturados. Existem vários algoritmos que pode trazer a quantidade de valores possíveis, escolher qual o melhor deles é uma tarefa delicada. Desta forma, a alternativa encontra-se **errada**.

**Gabarito: A****50. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI  
-QUESTÃO 3**

```
CREATE TABLE predio
(
    id numeric(7,0),
    nome varchar(50),
    local varchar(150),
    mnemonico varchar(10),
    CONSTRAINT pk_sede PRIMARY KEY (id),
    CONSTRAINT uq_sede UNIQUE (mnemonico)
);

CREATE TABLE salas
(
    codigo numeric(7,0) NOT NULL,
    local varchar(10),
    descricao varchar(50),
    area numeric(10,2),
    CONSTRAINT pk_salas PRIMARY KEY (codigo),
    CONSTRAINT fk_sede_sala FOREIGN KEY (local)
        REFERENCES predio (mnemonico)
);
```

Considerando os algoritmos acima, em que são criadas as tabelas predio e salas, assinale a opção cuja expressão SQL apresenta informações do registro da maior sala existente.

A select c1.local, c1.nome, c2.descricao, c2.area  
from predio as c1, salas as c2  
where c2.local=c1.mnemonico  
having max(c2.area)

B select c1.local, c1.nome, c2.descricao, max(c2.area)  
from predio as c1, salas as c2  
where c2.local=c1.id  
group by c1.local, c1.nome, c2.descricao

C select c1.local, c1.nome, c2.descricao  
from predio as c1, (  
select local, descricao, area from salas as c1  
where area = (select max(area) from salas as  
c2 where area>0)  
) as c2 where c2.local=c1.mnemonico;

D select c1.local, c1.nome, c2.descricao, c2.area  
from predio as c1, (  
select local, descricao, area from salas as c1  
where area = (select max(area) from salas as  
c2 where area>0)  
) as c2 where c2.id=c1.codigo;

E select c1.local, c1.nome, c2.descricao, max(c2.area)  
from predio as c1 join salas as c2  
on c2.codigo=c1.id  
group by c1.local, c1.nome, c2.descricao

**Comentário:** Vamos procurar achar os erros das alternativas distintas da resposta da questão, analisaremos, portanto, na ordem que aparece na questão.

A O erro desta alternativa está no uso da função agregada sem que apareça o group by. Vejam também que não temos nenhuma função agregada descrita na cláusula select, por fim, a sintaxe do comando "having max(c2.area)" está incorreta, seria necessário comparar o valor de max(coluna) com uma constante ou outra variável, por exemplo max(c2.area) > 200;

B Veja que na alternativa B existe um erro lógico na comparação "c2.local=c1.id", ela não faz sentido. Não traz para o resultado as tuplas necessárias.

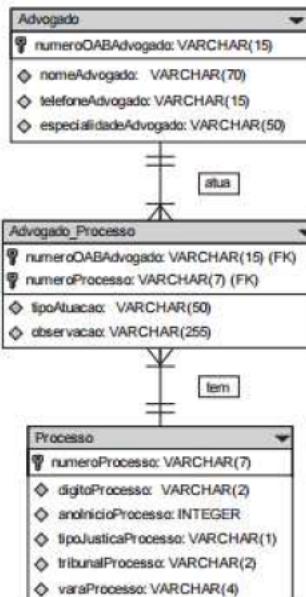
C A alternativa C é a nossa resposta. Veja que primeiro é feita uma consulta interna para saber qual a sala que tem a área maior. Depois, outra subconsulta retorna os atributos que fazem parte da tupla que possui a sala com a maior área. Num terceiro momento, na consulta externa, recuperamos a informação do prédio, baseado na comparação do atributo mnemônico que funciona como atributo de ligação ou chave estrangeira, que relaciona as duas tabelas.

D A letra D usa o mesmo raciocínio da C, mas peca ao considerar a chave estrangeira outro atributo (c2.id=c1.codigo) desta forma não é possível correlacionar as duas tabelas.

E A alternativa E também não retorna o resultado adequado, precisaríamos de uma restrição sobre a função de agregação para retornar apenas a linha que possuísse o valor máximo, isso poderia ser feito por meio do uso da cláusula having e de uma subconsulta que retornasse o valor máximo.

**Gabarito: C**

A figura abaixo se refere às questões 03 a 05.



Dados cadastrados nas tabelas:

numeroDABAAdvogado	nomeAdvogado	telefoneAdvogado	especialidadeAdvogado
13894	Marcela Teodoro	(17)9999-9999	Trabalhista
34001	Marco Aurélio Pereira	(18)8888-8888	Trabalhista
59445	Pedro Eduardo Silva	(11)7777-7777	Trabalhista
67812	Ana Maria Souza	(21)6666-6666	Trabalhista

numeroProcesso	digitoProcesso	anoinicioProcesso	tipoJusticaProcesso	tribunalProcesso	varaProcesso
000182	18	2010	5	03	0032
000346	01	2008	5	15	0054
000467	45	2010	5	05	0034
001367	10	2007	5	05	0012

numeroDABAAdvogado	numeroProcesso	tipoAtuacao	observacao
34001	000182	Defesa	Aguardando aprovação
34001	000346	Defesa	
34001	001367	Acusação	
59445	000346	Acusação	Alterado
59445	000467	Acusação	
59445	001367	Defesa	



### 51. ANO: 2015 BANCA: FCC ÓRGÃO: TRT - 3ª REGIÃO (MG) PROVA: TÉCNICO JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Para exibir o nome de todos os advogados que NÃO estão ligados a nenhum processo na tabela advogado\_processo utiliza-se a instrução:

A SELECT DISTINCT nomeAdvogado FROM advogado WHERE nomeAdvogado NOT IN (SELECT nomeAdvogado FROM advogado\_processo);

B SELECT nomeAdvogado FROM advogado WHERE numeroOABAdvogado IS NOT(SELECT numeroOABAdvogado FROM advogado\_processo);

C SELECT nomeAdvogado FROM advogado RIGHT JOIN advogado\_processo ON advogado.numeroOABAdvogado <> advogado\_processo.numeroOABAdvogado;

D SELECT nomeAdvogado FROM advogado WHERE numeroOABAdvogado NOT IN (SELECT numeroOABAdvogado FROM advogado\_processo);

E SELECT DISTINCT nomeAdvogado FROM advogado JOIN advogado\_processo ON advogado.numeroOABAdvogado = advogado\_processo.numeroOABAdvogado;

**Comentários:** O gabarito desta questão encontra-se na alternativa D. Ela apresenta o uso correto do NOT IN. Veja que para descobrir quais advogados não estão ligados a nenhum processo basicamente listamos os advogados e perguntamos quais deles não aparecem na lista da relação advogado\_processo.

Existem outras três palavras-chave que são bastante úteis quanto trabalhamos com subconsultas: são ALL, ANY, e SOME. Elas trabalham com operadores de comparação e conjuntos de resultados. Vejamos alguns exemplos começando pelo ALL.

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ALL
(SELECT rating FROM restaurant_ratings
WHERE rating > 3 AND rating < 9);
```

Neste caso, acima, estamos selecionando todos os nomes e rating de restaurantes cujo rating é maior do que todos os ratings da subconsulta. Vamos agora para o ANY.

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ANY
(SELECT rating FROM restaurant_ratings WHERE
rating > 3 AND rating < 9);
```

Agora verificamos se a variável rating é maior do que algum dos valores da subconsulta. Podemos substituir o ANY por SOME, ambas têm o mesmo significado.

**Gabarito D.**



## 52. BANCA: FCC ANO: 2015 ÓRGÃO: TRT - 3<sup>a</sup> REGIÃO (MG) PROVA: TÉCNICO JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Um técnico de TI com perfil autorizado para executar operações DML nas tabelas do banco de dados digitou um conjunto de instruções SQL, mas foi executada com sucesso apenas a instrução:

- A INSERT INTO advogado\_processo VALUES ('34001', '000467', 'Acusação', 'Aprovado');
- B INSERT INTO advogado\_processo (numeroOABAdvogado, numeroProcesso, tipoAtuacao) VALUES ('67812', '0001467', 'Acusação');
- C INSERT INTO advogado\_processo (numeroOABAdvogado, numeroProcesso, tipoAtuacao) VALUES ('59445', '000346', 'Acusação');
- D UPDATE advogado ALTER COLUMN telefoneAdvogado='(11)8787-8787' WHERE numeroOABAdvogado='67812';
- E UPDATE advogado\_processo SET numeroOABAdvogado='59800' WHERE numeroOABAdvogado='59445' and numeroProcesso='000467';

**Comentário:** Vejam que todas as alternativas apresentam erros exceto a resposta. Alguns erros estão relacionados ao fato de não existir a informação, por exemplo, na alternativa B não temos o numero do processo 0001467. Vejam que, neste caso, não será possível inserir por conta da integridade referencial. Para inserir um valor na tabela advogado\_processo, tanto o processo quanto o advogado precisam estar cadastrados.

O fato acontece na alternativa A, tanto o valor 34001 é um valor válido na tabela de advogado, quanto o valor 000467 é um valor presente na tabela de processo.

Na alternativa C, o técnico tentou inserir uma informação para um valor de chave primária já existente. Vejam que a sintaxe do comando UPDATE é incorreta na alternativa D. Por fim, a letra E, está tentando atualizar um advogado que não existe na base. Ficamos com a nossa resposta na letra A.

### Gabarito A.



### 53. BANCA: FCC ANO: 2015 ÓRGÃO: TRT - 3<sup>a</sup> REGIÃO (MG) PROVA: TÉCNICO JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Um técnico de TI criou uma view e executou a instrução SELECT \* FROM processos2010;, que exibiu apenas os dados a seguir:

Número do Processo	Número do Tribunal
000182	03
000346	15
000467	05

A instrução utilizada para criar a view foi:

A CREATE VIEW processos2010 AS SELECT numeroProcesso AS Número\_do\_Processo, tribunalProcesso AS Número\_do\_Tribunal FROM processo WHERE anoInicioProcesso>=2007;

- B CREATE VIEW processos2010 AS SELECT numeroProcesso Número\_do\_Processo, tribunalProcesso Número\_do\_Tribunal FROM processo WHERE anoInicioProcesso>2007;
- C CREATE VIEW processos2010 ON SELECT numeroProcesso Número\_do\_Processo, tribunalProcesso Número\_do\_Tribunal FROM processo WHERE tribunalProcesso<10;
- D CREATE VIEW processos2010 WITH SELECT numeroProcesso Número\_do\_Processo, tribunalProcesso Número\_do\_Tribunal FROM processo WHERE anoInicioProcesso>2009;
- E CREATE VIEW processos2010 AS SELECT numeroProcesso AS Número\_do\_Processo, tribunalProcesso AS Número\_do\_Tribunal FROM processo WHERE tribunalProcesso>=5; SELECT \* FROM processos2010;

**Comentário:** A questão pede para criarmos uma visão sobre as tabelas descritas no modelo presente na página anterior. Pelo enunciado podemos concluir que uma visão cujo nome é processos2010 será criar. As informações geradas pela execução da VIEW também podem ser vistas no enunciado da questão. Vejam que ele executou um SELECT \* que retorna todas as linhas e colunas da relação pesquisada.

Partindo das precisas acima, precisamos verificar qual a restrição sobre a tabela de processos nos retornaria as tuplas do enunciado. Para entender a questão, basta aplicar as restrições da cláusula WHERE de cada uma das alternativas sobre a tabela de processos e verificar se o resultado está adequado com o solicitado no enunciado.

Vejam que, embora ele tenha nomeado a view como processos2010, ele selecionou os registros referentes a processos "maiores do que 2007". Desta forma a alternativa B é a nossa resposta.

**Gabarito B.****54. BANCA: CESPE ANO: 2015 ÓRGÃO: FUB PROVA: ANALISTA ADMINISTRATIVO - ANALISTA DE TECNOLOGIA DA INFORMAÇÃO**

Julgue os itens seguintes, no que se refere à linguagem SQL.

91 Supondo que seja necessário buscar dados em duas tabelas distintas, o comando select não deve ser escolhido por não possuir os recursos para efetuar a busca em ambas as tabelas e exibir o resultado.

92 A função max, utilizada conjuntamente com o comando select, retorna o maior valor em um determinado campo que tenha sido incluído na busca.

**Comentário:** O primeiro item falha ao dizer que o comando select não possui recurso para buscar dados em duas tabelas distintas. Sabemos que a cláusula FROM pode ser usada para listar ou ainda relacionar diferentes tabelas. A forma

mais simples de usar diferentes colunas é apenas listando seus nomes separados por vírgula. Neste caso você vai forçar o SGBD a executar um produto cartesiano entre as relações passadas.

A outra opção de retornar no resultado dados de mais de uma relação seria por meio do comando JOIN, neste caso teríamos uma execução otimizada visto que os atributos de junção fariam uma restrição no resultado durante o processamento da consulta.

O item 92 trata da função agregada max() que de fato retorna o valor máximo de uma determinada coluna de uma tabela. Vamos aproveitar para falarmos um pouco mais sobre as funções de agregação.

Uma função de agregação recebe um conjunto de valores e retorna um valor de saída. Uma das funções de agregação mais comuns é COUNT, que conta valores não nulos em uma coluna. Por exemplo, para contar o número de cachoeiras associadas a um estado, especificar:

```
SELECT COUNT (u.county_id) AS county_count FROM upfall u;
```

É possível adicionar DISTINCT na consulta anterior para contar o número de municípios que contenham cachoeiras:

```
SELECT COUNT (DISTINCT u.county_id) AS county_count FROM upfall u;
```

O comportamento ALL é o padrão, ele conta todos os valores: COUNT (expressão) é equivalente a COUNT (ALL expressão). COUNT é um caso especial de uma função de agregação porque você pode passar o asterisco (\*) para contar a quantidade de linhas retornadas:

```
SELECT COUNT (*) FROM upfall;
```

A nulidade é irrelevante quando COUNT (\*) é usado porque o conceito de nulo se aplica apenas a colunas, não as linhas como um todo. Todas as outras funções agregadas ignoram valores nulos.

Abaixo apresentamos uma lista com algumas funções agregadas comumente usadas. No entanto, a maioria dos fornecedores de banco de dados implementam funções de agregação além das mostradas.

#### Função - Descrição

AVG (x) - Retorna a média.

COUNT (x) - Conta os valores não nulos.

MAX (x) - Retorna o maior valor.

MEDIAN (x) - Devolve a mediana

MIN (x) - Retorna o menor valor.

STDDEV (x) - Retorna o desvio padrão.

SUM (x) - Resume todos os números.

VARIANCE (x) - Retorna a variância estatística.

**Gabarito: E C****55. BANCA: UERJ ANO: 2015 ÓRGÃO: UERJ PROVA: ANALISTA DE SISTEMAS - SUPORTE E INFRAESTRUTURA**

A solução correta para que uma consulta sql retorne às agências que possuem média dos saldos aplicados em conta maior que 1200 é:

A select nome\_agencia, avg(saldo) from conta group by nome\_agencia having avg(saldo) > 1200

B select nome\_agencia, avg(saldo) from conta where ( having avg(saldo) > 1200 )

C select nome\_agencia, avg(saldo) from conta where avg(saldo) > 1200 group by nome\_agencia

D select nome\_agencia, avg(saldo) from conta group by nome\_agencia having saldo > 1200

**Comentário:** Observem que a questão tenta verificar seu entendimento sobre a sintaxe correta do comando SELECT quando utilizamos uma função de agregação e queremos fazer uma restrição sobre o resultado dessa função. Vejam que basicamente usamos a mesma sintaxe antecedida pelo having. Desta forma podemos fazer restrições sobre o resultado de uma função de agregação.

Podemos ainda observar que os atributos presentes na cláusula SELECT que não estão dentro da função de agregação devem constar na cláusula GRUOP BY.

**Gabarito A.****56. BANCA: UERJ ANO: 2015 ÓRGÃO: UERJ PROVA: ANALISTA DE SISTEMAS - SUPORTE E INFRAESTRUTURA**

O sistema de linguagem SQL (Structured Query Language, ou Linguagem de Consulta Estruturada) possui a seguinte característica:

A está centrado no conceito de adição ou remoção de servidores de banco de dados de forma escalável e sem interrupção dos serviços

B está baseado no relacionamento estruturado para endereços de nomes de domínios com seus respectivos endereços de máquina

C está baseado na pesquisa declarativa padrão para banco de dados relacional; muitas de suas características foram inspiradas na álgebra relacional

D está centrado em bancos de dados distribuídos, onde dados são armazenados em múltiplos pontos de processamento e normalmente em muitos servidores

**Comentário:** A questão é interessante por tratar dos conceitos relacionados a linguagem SQL. Vejam que ela é uma linguagem declarativa, nela você não precisa descrever como o SGBD vai executar a consulta. Outra importante característica é a base matemática do modelo, baseado na álgebra relacional. Sendo assim podemos perceber nossa resposta na alternativa C.

As demais alternativas estão totalmente equivocadas.

**Gabarito C.****57. BANCA: UERJ ANO: 2015 ÓRGÃO: UERJ PROVA: ANALISTA DE SISTEMAS - DESENVOLVIMENTO**

Seja o esquema relacional apresentado a seguir:

dvd = (código, nome, data\_da\_compra, preço, região)

Para mostrar os campos nome e data\_da\_compra dos dvds, classificados por data de compra em ordem decrescente, deve-se usar a instrução SQL:

- A Select nome, data\_da\_compra From dvd Inverse Order By data\_da\_compra
- B Select nome, data\_da\_compra From dvd Order By data\_da\_compra not Asc
- C Select nome, data\_da\_compra From dvd Order By data\_da\_compra Desc
- D Select nome, data\_da\_compra From dvd Order By data\_da\_compra Group By Down

**Comentário:** A questão é interessante para analisar ou recordarmos o fato da ordenação feita pela cláusula ORDER BY é ascendente por default. Veja que por conta disso precisamos especificar, por meio do uso do DESC, que estamos solicitando a ordenação uma ordenação decrescente. Se quisermos ordenar de forma crescente não precisaríamos inserir nenhuma cláusula.

**Gabarito C.****58. BANCA: IESES ANO: 2015 ÓRGÃO: IFC-SC PROVA: INFORMÁTICA - WEB DESIGN**

Sobre SQL, assinale a alternativa correta:

- A GRANT é uma instrução do tipo DML.
- B DROP é uma instrução do tipo DML.
- C CREATE INDEX é uma instrução do tipo DML.

D UPDATE é uma instrução do tipo DML.

**Comentário:** Questão que aborda as diferentes linguagens que subdividem SQL. Temos as tradicionais DDL e DML, para definição e manipulação de dados, respectivamente. Como exemplo de DDL temos: CREATE, ALTER, DROP e TRUNCATE. Pelo lado da DML temos: SELECT, INSERT, UPDATE e DELETE. Se quisermos podemos ainda inserir dentro deste contexto a linguagem de controle de dados – DCL, composta pelos comandos GRANT e REVOKE. Vejam a figura abaixo com a lista de comandos e suas descrições:

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query

#### Gabarito D.



#### 59. BANCA: IESES ANO: 2015 ÓRGÃO: IFC-SC PROVA: INFORMÁTICA - BANCO DE DADOS

Considere as tabelas a seguir, onde “cod\_cliente” na tabela “Compras” é uma chave estrangeira para o campo “cod\_cliente” da tabela “Clientes”:

Tabela_Compras		
cod_cliente	data_compra	valor_compra
1	01/04/2015	100,00
4	02/04/2015	99,00
2	04/04/2015	57,80
1	31/03/2015	66,90
4	15/04/2015	33,40
2	25/03/2015	25,00

Tabela_Cliente	
cod_cliente	nome_cliente
1	José
2	Pedro
3	Andrea
4	Adolfo

Qual seria o resultado da consulta SQL

"select sum(Tabela\_Compras.valor\_compra)  
from Tabela\_Compras where  
Tabela\_Compras.data\_compra between "01/04/2015" and "15/04/2015" and  
Tabela\_Compras.cod\_cliente = 2",  
e executado no esquema acima?

A 290,20.

B 357,10.

C 99,00 e 33,40.

D 57,8

**Comentário:** A questão ficou fácil quando o examinador restringiu a busca por meio do texto Tabela\_Compras.cod\_cliente = 2 na cláusula WHERE. Contudo, temos um comentário relevante para fazer sobre o comando between.

O operador BETWEEN permite que você especifique um intervalo, entre dois valores. Quando você precisa procurar um valor dentro de um determinado intervalo, você usa o operador "maior ou igual a" ( $\geq$ ) ou o operador "menor ou igual a" ( $\leq$ ).

O operador BETWEEN funciona exatamente da mesma maneira, exceto que é mais curto, economiza digitação e também faz com que o SQL seja mais legível. O seguinte SQL usa o operador BETWEEN para selecionar filmes com uma classificação entre 3 e 5:

```
SELEÇÃO FilmName, Avaliação
FROM filmes
WHERE rating BETWEEN 3 AND 5;
```

**Gabarito D.**



#### 60. BANCA: IESES ANO: 2015 ÓRGÃO: IFC-SC PROVA: INFORMÁTICA - BANCO DE DADOS

Supondo que em dado banco de dados existe uma tabela chamada "usuario", onde foi identificada a necessidade de criação de um índice chamado "Idx\_estado" para o campo "cod\_estado". Qual das alternativas apresenta o comando SQL correto?

A ON TABLE `usuario` ADD INDEX `Idx\_estado` ( `cod\_estado` );

B ALTER TABLE `usuario` INDEX ADD `Idx\_estado` ( `cod\_estado` );

C ALTER TABLE `usuario` ADD INDEX `Idx\_estado` ( `cod\_estado` );

D CREATE INDEX IN `usuario` on `Idx\_estado` ( `cod\_estado` );

**Comentário:** Por meio do comando ALTER TABLE é possível, além de inserir ou modificar colunas, adicionar índices e restrições de integridade.

**Gabarito C.****61. ANO: 2015 BANCA: NUCEPE ÓRGÃO: SEFAZ - PI PROVA: ANALISTA - SISTEMAS PLENO**

Qual a interpretação para o seguinte comando da linguagem SQL?

```
Select A.Nome  
  From Alunos A, Historico H  
 Where A.RA = H.RA  
   and ano = 2014  
   and nota < 7.0  
 group by A.Nome  
 having count(*) >= 3
```

A Verifica se todos os alunos tiveram no mínimo 3 (três) notas abaixo de 7 (sete) em 2014.

B Verifica se há algum histórico com no mínimo 3 (três) notas abaixo de 7 (sete) em 2014.

C Fornece os nomes dos alunos que tiveram no mínimo 3 (três) notas abaixo de 7 (sete) em 2014 em seu histórico, agrupados por nota.

D Fornece as notas dos alunos que tiveram no mínimo 3 (três) notas abaixo de 7 (sete) em 2014 em seu histórico.

E Fornece o nome dos alunos que tiveram no mínimo 3 (três) notas abaixo de 7 (sete) em 2014 em seu histórico.

**Comentário:** O comando realiza uma junção entre alunos e histórico, recuperando o nome dos alunos que tiveram notas menores do que sete em 2014. A cláusula having limita os resultados a somente os alunos que tiveram três ou mais notas menores que sete.

**Gabarito E.****62. BANCA: NUCEPE ANO: 2015 ÓRGÃO: SEFAZ - PI PROVA: ANALISTA - SISTEMAS JÚNIOR**

Qual o resultado obtido com o comando SQL abaixo?

"SELECT tipo\_vinho, MAX (preço), AVG (preço) FROM vinho GROUP BY tipo\_vinho HAVING AVG (preço) > 200"?

A Retorna o preço mais alto e a média dos preços por tipo de vinho, para médias de preços inferiores a R\$200,00.

B Retorna o preço mais alto e a média dos preços por tipo de vinho, para médias de preços superiores a R\$200,00.

C Retorna a soma dos preços e a média dos preços por tipo de vinho, para médias de preços superiores a R\$200,00.

D Retorna o preço mais alto e o menor dos preços por tipo de vinho, para médias de preços superiores a R\$200,00.

E Retorna a soma dos preços mais altos e a média dos preços por tipo de vinho, para médias de preços iguais a R\$200,00.

**Comentário:** Mais uma questão que mostra a utilização das funções agregadas dentro do comando SELECT. Vejam que são feitos dois agrupamentos e é utilizada a cláusula *having* para fazermos restrições sobre a média do preço.

**Gabarito B.****63. BANCA: CESPE ANO: 2008 ÓRGÃO: TJ-DF PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO**

Quanto a bancos de dados, sistemas gerenciadores de bancos de dados e técnicas correlacionadas de modelagem de dados, julgue os próximos itens.

Na linguagem de consulta SQL (structured query language), é possível obter o resultado de uma consulta SELECT ordenado pelo valor de um ou mais atributos.

**Comentário:** Sim! Basta utilizar a cláusula ORDER BY e as opções ASC ou DESC, para ordenação, ascendente ou descendente, respectivamente.

**Gabarito C.****64. BANCA: MP-RS ANO: 2015 ÓRGÃO: MP-RS PROVA: TÉCNICO EM INFORMÁTICA - INTERNET/INTRANET**

Considere a seguinte consulta a ser executada em um banco de dados MySQL.

```
select count(distinct job) from employee;
```

Essa consulta informa

A quantos valores diferentes existem na coluna job.

B o número de funcionários que possuem um job.

- C o número de funcionários agrupados por job.  
D os nomes dos jobs em ordem alfabética inversa.  
E o número total de funcionários da tabela employee.

**Comentário:** Nesta questão observamos o uso da cláusula `distinct`, que é utilizada para eliminar tuplas duplicadas do resultado, final ou intermediário, de uma consulta. Sendo assim, o comando vai permitir contar quantos valores diferentes existem na coluna `job`.

**Gabarito A.****65. BANCA: MP-RS ANO: 2015 ÓRGÃO: MP-RS PROVA: TÉCNICO EM INFORMÁTICA - SISTEMAS**

Assinale com V (verdadeiro) ou com F (falso) as seguintes afirmações, relativas à Linguagem de Definição de Dados (DDL) em SQL.

- ( ) `UNIQUE` indica que não pode haver repetição no conteúdo da coluna.  
( ) `NOT NULL` indica que o conteúdo da coluna não pode ser alterado.  
( ) `PRIMARY KEY` permite atribuir um conteúdo padrão a uma coluna da tabela.  
( ) `CHECK` permite especificar quais valores podem ser utilizados para preencher a coluna.

A sequência correta de preenchimento dos parênteses, de cima para baixo, é

- A V – F – V – F.  
B V – F – F – V.  
C F – V – F – V.  
D F – V – V – F.  
E F – F – V – V.

**Comentário:** A questão trata de aspectos referentes a restrições de integridade. Quando usamos a palavra reservada `UNIQUE`, estamos limitando os valores de uma coluna, impedindo que valores duplicados sejam inseridos. O `NOT NULL` não permite que uma determinada coluna ou atributo de uma tabela armazene valores nulos.

Quando utilizamos `PRIMARY KEY` estamos determinando qual a chave primária de uma tabela. Por fim o atributo `CHECK` serve para validar um determinado atributo diante de um conjunto de restrições previamente definidas.

**Gabarito B.**

## Considerações finais

Chegamos, pois, ao final da nossa aula de SQL!

Espero que você esteja curtindo o assunto de banco de dados. Cada passo dentro do assunto deve ser dado com segurança, portanto, não deixe de mandar suas dúvidas por um dos canais de atendimento do Estratégia.

Espero que tenha gostado! E até a próxima aula!

As videoaulas de SQL já estão disponíveis no site. Peço que depois de lerem esse material, assistam aos vídeos e tentem esgotar todas as questões sobre o assunto de algum site de questões. Qualquer dúvida pode postar no fórum aqui do Estratégia.

Gostaria de agradecer a todos você que adquiriram o material do Estratégia Concursos.

Forte abraço

Thiago Cavalcanti

## Referências

Fiz uma lista com alguns links de referências caso você quisesse se aprofundar um pouco.

- i. SQL in 24 Hours, Sams Teach Yourself, Sixth Edition
- ii. Beginning SQL, By: Paul Wilton; John W. Colby, Publisher: Wrox
- iii. SQL For Dummies, 8th Edition